우리식조작체계문고

Shell

교육성교육정보쎈터
주체97(2008)

# 차 례

# 머 리 말

위대한 령도자 **김정일**동지께서는 다음과 같이 지적하시였다.

**《프로그람을 개발하는데서 기본은 우리 식의 프로그람을 개발하는것입니다. 우리는 우리 식의 프로그람을 개발하는 방향으로 나가야 합니다.》**(《김정일선집》 제15권, 196페지)

위대한 령도자 **김정일**동지께서는 나라의 정보산업을 보다 발전시키기 위해서는 우리식의 조작체계에 의한 프로그람을 개발하는것이라고 가르쳐주시였다.

지금 세계적으로 자기식의 조작체계를 개발하는데 많은 관심을 돌리고있다.

그중에서도 Linux에 대한 관심이 더욱 더 커가고있는것이 최근시기의 추세이다. Linux는 Windows와 달리 원천코드가 공개되여있고 강력한 보안과 망봉사기능을 가지고있다.

Linux의 이러한 특징은 오늘 콤퓨터망전문가들과 관리자들이 Linux조작체계에 대하여 보다 큰 관심을 가지게 하고있으며 사용자들의 리용률이 높아지고있다.

우리식 조작체계는 이러한 리눅스를 핵심부로 하여 그것을 우리나라의 실정에 맞게 완성하였다. 그러므로 우리식 조작체계를 잘 알기 위해서는 Linux조작체계에 대하여 잘 알아야 한다.

이 책에서는 Linux의 기반으로 되고있는 쉘프로그람작성과 프로그람개발언어의 하나인 perl에 대하여 서술하고있다.

쉘은 Linux에서 핵심부와 프로그람들을 련결하여주는 스크립트의 하나로써 Linux 사용자가 반드시 알아야 프로그람언어이다.

체계의 설정파일들과 기동스크립트들은 쉘로 작성되여있다. 쉘프로그람작성에 대하여 알면 Linux를 GUI방식에서의 설정파일을 변경시키는 조작으로 자기 환경을 변경시키는것뿐아니라 자기만의 특수한 환경을 조종탁상에서 실현시킬수 있으며 마우스로 하기 힘든 조작을 간단한 스크립트로 쉽게 작성할수 있다.

Linux는 사용자가 부단히 창조하고 혁신할것을 권고하고있다. 지령들을 결합하여 쓰거나 복잡한 일감을 수행하는 스크립트를 설계하는것은 Linux사용자들에게 있어서 실제적인 과제로 된다. 또한 체계에 있는 프로그람들을 조작하는데서도 쉘언어를 잘 아는것이 필요하다.

이러한 요구를 반영하여 이 책에서는 Linux에서 기본쉘로 되고있는 bash쉘에 대하여 구체적인 실례를 들어 설명하였다.

이 책이 Linux쉘을 배우려는 독자들에게 도움이 되기를 바란다.

# 제1장. 쉘이란 무엇인가

## 제1절. 쉘의 개념

### 1.1.1. 쉘에 대한 정의

**쉘이란 무엇인가?**

쉘은 사용자와 조작체계핵심부(Kernel)사이의 대면부를 제공해주는 일종의 특수한 프로그람언어이다. 간단히 말하여 쉘은 사용자가 입력한 지령을 해석하고 실행하는 지령해석기이다.

쉘이 핵심부와 사용자사이의 대면부를 담당했으므로 일부 사용자들은 쉘 그 자체를 조작체계로 리해하는 경우도 있지만 쉽게 말하여 지금까지 오래동안 사용해온 MS-DOS와 비슷한 역할을 한다고 볼수 있다.

사용자가 조작체계를 습득한다는것은 쉘에서 제공하는 각종 지령들의 사용법을 학습하는것이라고 말할수 있다. 사용자가 조작체계에 접근하여 지령을 주고받자면 쉘의 도움을 받아야 한다.

이처럼 조작체계에서 쉘은 매우 중요한 지위에 있다.(그림 1-1)

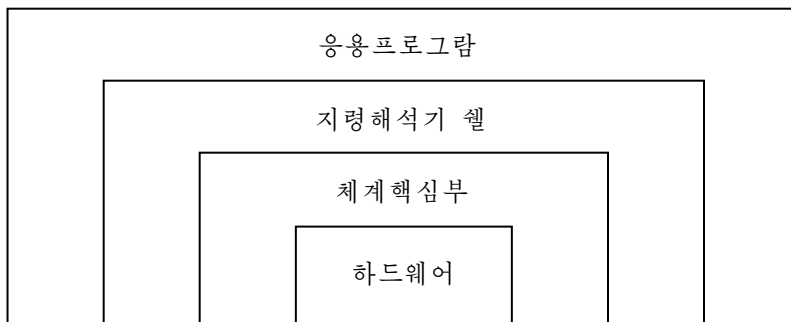



그림 1-1. 조작체계에서 쉘의 지위

### 1.1.2. 쉘의 기능과 역할

**1. 쉘의 기능**

△ 쉘의 중요한 기능의 하나는 입력된 지령들을 해석하는 문자사용자대면부를 사용한 대화형기능이다.

쉘은 지령행의 구문을 해석하고 타브, 빈칸, 행바꾸기로 구분되는 단어들(일명 통표: token)로 분리한다. 만일 단어들이 특정한 메타문자들을 포함하면 쉘은 그 메타문자들을 해석한다.

**메타문자(meta charater)**

\, *, ?, [ ], { }, !, $ 등과 같이 원천프로그람이나 자료파일에 들어있으면서 그 자체가 문자를 나타내는것이 아니라 다른 문자에 대한 정보나 의미를 표현하는 특수문자, 실례로 $는 그 다음에 오는 변수의 치환을 의미한다.

```
Name=kim
echo "$Name"
```

우의 실례에서 Name이 아니라 kim이라는 문자렬이 현시된다.

그 다음 파일입출력과 배경프로쎄스를 처리한다. 지령행에 지령이 정확히 입력되였을 때 쉘은 지령을 해석하고 실행한다.

그러면 문자사용자대면부를 왜 사용하는가?

Linux개발초기에 이 조작체계에는 도형사용자대면부(GUI: Graphical User Interface)가 없었고 오직 문자사용자대면부만이 있었다. 물론 지금은 Linux조작체계에도 X Window라는 GUI가 개발되여있다. GUI조작이 아무리 편리하다해도 문자사용자대면부는 다음과 같은 우점으로 하여 지금도 여전히 사용되고있다.

① 체계관리를 비롯한 일련의 과제를 효과적으로 수행할수 있다.

체계관리는 보통 원격으로 진행되는 경우가 많은데 원격등록가입은 문자사용자대면부로 이루어진다.

② 체계자원을 극력 절약하여 빠른 속도를 보장한다.

다음의 3가지 방법으로 문자사용자대면부로 들어갈수 있다.

- 도형사용자대면부에서 조종탁창문을 열어 문자사용자대면부로 들어간다.
- 체계기동을 문자사용자대면부방식으로 한다.
- telnet와 ssh를 리용한 원격등록가입방식으로 문자사용자대면부로 들어간다.

체계기동시에 문자사용자대면부로 등록가입하면 체계는 여러개(기정으로 6개)의 가상조종탁을 제공한다. 매개 가상조종탁은 독립적으로 사용되며 서로 영향을 주지 않는다. Alt+F1～Alt+F6을 사용하여 가상조종탁사이를 절환할수 있다.

사용자가 startx지령에 의하여 문자사용자대면부에서 도형사용자대면부로 기동했으면 Ctrl+Alt+F1～Ctrl+Alt+F6을 리용하여 가상말단사이를 절환할수 있다. Ctrl+Alt+F7을 사용하면 도형사용자대면부로 넘어간다. 만일 사용자가 체계기동후에 직접 문자사용자대면부로 들어가든가 혹은 원격으로 문자사용자대면부로 들어가면 다음과 같은 화면을 보게 될것이다.

```
《붉은별》 (사용자용체계 1.0판)
핵심부 2.6.18-1.2798 구성방식 i686

localhost login:
```

그림 1-2. 문자사용자대면부에서 Linux로 등록가입

만일 등록탈퇴하려면 logout지령을 사용하든가 혹은 Ctrl+D지령을 사용하면 된다. 특권사용자(root)의 재촉문(prompt)기호는 #이고 일반사용자의 재촉문기호는 $이다.

△ 쉘의 중요한 기능은 사용자의 환경을 전용화하는 기능이다.

이것은 보통 쉘초기화파일에서 진행된다. 이 파일에는 말단건들과 창문속성, 탐색경로, 재촉문과 말단형태를 정의하는 변수들, 창문, 본문편집프로그람, 프로그람작성언어를 위한 서고 등에 대한 변수들의 설정이 들어있다. Korn쉘과 C쉘은 사용자가 파일을 지우거나 부주의로 등록탈퇴하지 못하게 하며 과제가 끝났을 때 사용자에게 그것을 알려주도록 설정되여있고 내부변수들과 전용화를 위한 리력과 별명들의 추가적인 기능을 제공한다.

△ 쉘은 해석프로그람작성언어로서 사용할수도 있다.

vi, emacs 등과 같은 편집기에서 작성되는 쉘프로그람(스크립트)은 변수대입, 조건검사, 순환명령과 같은 기초적인 프로그람작성구조들을 적절히 배합한 Linux지령들로 구성된다. 쉘은 스크립트를 완전히 작성한 다음 콤파일하지 않고 스크립트의 매개 행을 건반으로 입력한 지령처럼 해석한다. Linux체계에서 거의 모든 실행파일은 쉘지령에 의하여 집행되기때문에 사용자는 해당 쉘지령의 용도와 파일의 종류를 잘 알아야 한다. 파일의 종류는 다음과 같다.

표 1-1. Linux체계에서 실행파일의 종류

| 종 류 | 설 명 |
|---|---|
| Linux지령 | /bin과 /sbin등록부에 있는 지령 |
| 내부지령 | 효률적으로 쓰기 위해서 쉘내부에 있는 일부 상용지령 |
| 편의프로그람 | /usr/bin, /usr/sbin, /usr/share, /usr/local/bin 등의 등록부에 있는 편의프로그람 혹은 도구 |
| 사용자프로그람 | 콤파일하여 실행파일을 만든 후에 쉘지령으로 실행할수 있는 C, C++언어로 작성된 *.c 나 *.cpp형태의 프로그람 |
| 쉘스크립트 | 쉘언어로 작성한 일괄처리파일 |

사용자가 어떤 지령을 입력하면 쉘은 우선 그것의 지령형태를 판단하여 내부지령이면 기억기에서, 외부지령 혹은 실행프로그람이면 하드디스크에서 해당한 체계기능을 호출하여 그것을 해석하고 실행한다.

지령을 탐색할 때 다음의 2가지 경우가 있을수 있다.

**- 사용자가 경로를 제시할 때**

쉘은 사용자가 제시하는 경로에 따라 탐색하고 그것이 있는가 없는가 하는 정보를 제시한다.

**- 사용자가 경로를 지정하지 않았을 때**

쉘은 환경변수 PATH에 의해 정해지는 경로를 따라 검색하고 지령이 있는가 없는

가 하는 정보를 제시한다.

다음의 그림에 쉘의 지령해석과정을 보여주었다.

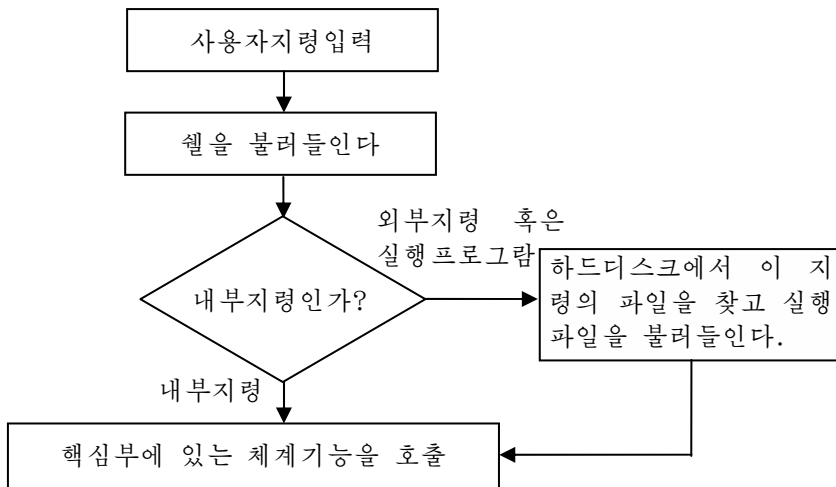



**그림 1-3. 쉘의 지령해석과정**

△ 쉘은 이밖에도 지령치환기능 즉 지령보충기능, 별명실행, 지령리력기능을 가지고 있다.

이에 대하여서는 2장 5절에서 구체적으로 설명한다.

**2. 쉘의 역할**

쉘의 역할은 한마디로 재촉문에서 입력되는 임의의 지령들이 정확히 실행되도록 하는것이다. 구체적으로 본다면 다음과 같다.

① 사용자가 입력한 지령을 읽고 지령행의 구문을 해석한다.(1.2.2. 참고)
② 메타문자를 포함한 특수문자들을 평가한다.(1.2.5. 참고)
③ 파이프, 방향바꾸기, 배경프로쎄스를 설정한다.(3장 2절 참고)
④ 신호를 처리한다.(3.6.3. 참고)
⑤ 프로그람을 정확히 실행시킨다.

## 1.1.3. 쉘의 종류

**1. 기본 UNIX쉘**

대부분의 UNIX체계가 제공하는 기본 쉘은 Bourne쉘, C쉘, Korn쉘이다. 이 3개의 쉘들은 사용자와 대화형으로 실행될 때에는 거의 같은 방식으로 동작하지만 스크립트작성언어로서 사용될 때에는 문법과 기능에서 일련의 차이가 있다.(표 1-2)

표 1-2. 기본 UNIX쉘

| 종　류 | 차　이 |
| --- | --- |
| Bourne쉘<br>(AT&T쉘) | 개 발 자: AT&T의 Stephen Bourne<br>개발년도: 1979년말<br>특징: 표준 UNIX쉘이며 초기에 주로 체계관리작업의 자동조종을 위해 리용되였다. 프로그람작성언어로서의 Bourne쉘은 알골(Algol)언어에 기초하고있다. stop, shutdown과 같은 대부분의 체계관리스크립트들은 Bourne쉘스크립트이고 단일사용자방식의 뿌리(root)에서 동작하는 체계관리자용 쉘도 Bourne쉘이다. 이 쉘은 간결하고 용량이 작으며 처리속도가 빠른것으로 인기가 있었지만 리력, 별명, 일감조종과 같은 대화형사용을 위한 많은 기능들이 없었다. Bourne쉘의 기정재촉문은 화페기호($)이다. |
| C쉘<br>(Berkeley쉘) | 개 발 자: Berkeley에 있는 캘리포니아종합대학의 Bill Joy<br>개발년도: 1970년대후반기<br>특징: 표준 Bourne쉘에서 제공되지 않은 지령행리력, 별명, 내부산수연산, 파일이름완성, 일감조종 등과 같은 일련의 기능들이 추가되였다. C쉘은 C언어에 기초하였기때문에 그것과 류사한 형식을 리용한다. 대화형사용측면에서는 C쉘이 Bourne쉘보다 더 좋지만 체계관리측면에서는 Bourne쉘스크립트가 C쉘스크립트보다 더 간결하고 빠르기때문에 주로 Bourne쉘을 사용한다. C쉘은 대형콤퓨터에서 설계되고 일련의 추가적인 기능들이 보충되였기때문에 소형콤퓨터에서는 속도가 뜨며 대형콤퓨터에서도 Bourne쉘보다 속도가 뜨다. C쉘의 기정재촉문은 퍼센트기호(%)이다. |
| Korn쉘<br>(Bourne쉘의 웃준위쉘) | 개 발 자: AT&T의 David Korn<br>개발년도: 1980년대 중엽<br>특징: Bourne쉘의 웃준위인 Korn쉘은 UNIX체계에서뿐아니라 OS/2, VMS, DOS에서도 실행된다. 이 쉘은 Bourne쉘과 웃방향호환성을 가지므로 이전의 Bourne쉘프로그람도 이 쉘에서 실행할수 있다. C쉘의 많은 기능들을 보충하였으며 C쉘보다 더 강력한 기능들인 편집할수 있는 리력, 별명, 함수들, 정규표현통용기호(regular expression wildcard), 내부산수연산, 일감조종, 협동프로쎄스, 특정한 오유수정수단 등이 추가되여 다른 쉘보다 빠르고 효률적이다. Korn쉘은 일련의 수정을 통하여 확장되였다. 비록 1993년판본이 인기를 모으고있지만 가장 광범히 사용되는 Korn쉘판본은 1988년판본이다. Korn쉘의 기정재촉문은 화페기호($)이다. |

## 2. Linux쉘

UNIX의 일종인 Linux체계가 지원하는 쉘에는 Bash쉘, TC쉘, Z쉘, 공개령역 Korn쉘 등 여러가지가 있다. 이중에서 가장 인기있는것이 Bash와 TC쉘이다. 그래서 이 쉘들을 흔히 《Linux쉘》이라고 부르지만 Solaris 8과 Sun UNIX 등 여러 UNIX 체계에서도 무료로 자유롭게 사용할수 있다.

① Bash쉘(Bourne Again Shell)

Brain Fox에 의해 개발되였으며 Linux조작체계에서는 표준쉘이다.

Bourne쉘에 기초하여 그 특성을 보충한것이며 C쉘과 Korn쉘의 우점을 포함하고 있다. Bourne쉘에 비해 개선된 기능은 지령행리력과 편집기능, 등록부탄창, 일감조종, 함수, 별명, 배렬, 옹근수산수연산기능이며 Korn쉘에서 제공하는 확장된 메타문자, 차

림표를 작성하기 위한 선택문, let지령 등과 같은 기능들도 추가되였다. Bash쉘은 강력한 프로그람작성기능을 가지고있으며 편리한 사용자대면부를 가지고있다.

② TC쉘(tcsh)

C쉘의 확장판본이다. 일련의 새로운 기능들로서는 emacs와 vi편집기를 리용한 지령행편집, 리력목록검색, 파일이름과 변수, 지령의 완성기능, 철자맞춤, 일감일정작성, 자동적인 자물쇠채우기, 자동등록탈퇴, 리력목록의 시간표시 등을 들수 있다.

③ Z쉘

Bash쉘, TC쉘, Korn쉘의 일부 기능들을 통합한 또 다른 Linux쉘이다.

④ 공개령역 Korn쉘(Public Domain Korn Shell)

일명 pdksh라고도 부르는 이 쉘은 Linux사용자들이 무료로 리용할수 있는 Korn쉘의 1988년판본이다.

자신이 사용하는 Linux체계가 어떤 쉘들을 지원하는가를 알려면 파일 /etc/shell을 조사하여야 한다. /etc/shell에 등록된 쉘중의 어느 하나를 리용하려면 chsh지령과 쉘의 이름을 입력해야 한다. Bash쉘을 리용하려면 재촉문에서 다음과 같은 형식의 지령을 입력하여야 한다.

```
chsh /bin/bash
```

## 제2절. 지령형식과 통용기호

### 1.2.1. 지령형식

쉘지령의 일반적인 형식은 다음과 같다.

```
지령이름 [-추가선택] [인수]
```

가장 간단한 쉘지령은 지령이름만 있고 복잡한 쉘지령은 여러개의 추가선택과 인수를 가지고있다. 추가선택과 인수들은 모두 쉘지령과 함께 입력하며 그것들사이는 공백으로 구분된다.

```
$ls
$ls -lra /home
$cat abc xyz
```

이 실례에서 ls와 cat는 지령이름이며 -l은 추가선택이고 /home은 인수이다.

사용자와 쉘사이 대화의 기본단위인 우와 같은 형식의 문자렬을 지령행이라고 부른다.

### 1.2.2. 지령행의 구문해석

재촉문에서 지령을 입력하면 쉘은 입력행을 읽고 통표라는 단어로 행을 분리하면서 지령행의 구문을 해석한다. 통표는 공백과 타브에 의해 분리되며 지령행은 행바꾸기에 의해 완료된다. 쉘은 첫 단어가 내부지령인가 디스크상의 실행프로그람인가를 검사하여 내부지령이면 내부적으로 지령을 실행하고 그렇지 않으면 프로그람이 어디에 있는가를

알기 위해 경로변수에 기입된 등록부를 탐색한다. 지령을 찾으면 쉘은 새로운 프로쎄스를 생성하고 프로그람을 실행한다. 쉘은 프로그람의 실행이 완료될 때까지 대기하며 필요하면 프로그람의 탈퇴상태를 알린다. 이때 재촉문이 나타나며 새 지령을 입력할수 있다.

지령행의 작업처리순서는 다음과 같다.

① 리력기능을 리용하여 지령치환을 진행한다.
② 지령행을 통표단위로 분리한다.
③ 리력을 갱신한다.
④ 인용부호를 처리한다.
⑤ 치환할 별명이 있는지 검사하고 함수들을 정의한다.
⑥ 방향바꾸기, 배경, 파이프 등을 설정한다.
⑦ 변수치환을 진행한다.
⑧ 지령치환을 진행한다.
⑨ 파일이름치환을 진행한다.
⑩ 프로그람을 실행한다.

### 1.2.3. 등록부와 파일이름

#### 1. 이름짓기규칙

Linux에서는 파일이름과 등록부이름을 작성할 때 다음의 규칙을 지켜야 한다.

기호 /를 내놓고 모든 기호를 쓸수 있다.

다음의 건과 문자들은 쓰지 않는것이 좋다.

공백건, 타브건, 후퇴건(backspace)과 문자부호들인 ? @ # $ & () \ | ; ' " < > 등이며 +와 -는 보통 이름의 첫문자로 사용하지 말아야 한다.

대소문자를 구별해야 한다.

#### 2. 파일확장자와 파일형태

DOS나 Windows환경에서 파일이름의 확장자는 파일의 형태를 가리킨다. 실례로 *.exe파일은 실행파일을, *.bat파일은 묶음파일을 가리킨다. Linux에서는 파일이 실행가능한 속성을 가지면 파일이름과 확장자에는 관계없이 실행할수 있다. 다만 일부 자료파일들에 확장자가 붙어있다.(표 1-3)

표 1-3. Linux에서 파일의 확장자

| 분 류 | 확장자이름 | 설 명 |
|---|---|---|
| 체계파일 | *.conf | 구성파일 |
| | *.rpm | RPM패키지 |
| | *.a | 일종의 보관파일 |
| | *.lock | 잠금파일 |
| | *~ | 여벌파일 |
| | .* | 숨은 파일 |

표계속

| 분 류 | 확장자이름 | 설 명 |
|---|---|---|
| 프로그람과 스크립트 | *.c | C언어의 원천프로그람파일 |
| | *.cpp | C++언어의 원천프로그람파일 |
| | *.h | C나 C++파일의 머리부파일 |
| | *.o | 프로그람객체파일 |
| | *.pl | perl언어의 원천프로그람파일 |
| | *.php | php언어의 원천프로그람파일 |
| | *.tcl | TCL스크립트파일 |
| | *.so/*.lib | 서고파일 |
| | *.sql | sql언어파일 |
| 격식파일 | *.txt | 본문파일 |
| | *.html/*.htm | 정적웨브페지파일 |
| | *.ps | PostScript파일 |
| | *.au | 음성파일 |
| | *.wav | 음성파일 |
| | *.xpm | 도형, 그림파일 |
| | *.jpg | 도형, 그림파일 |
| | *.gif | 도형, 그림파일 |
| | *.png | 도형, 그림파일 |
| 보존 및 압축파일 | *.tar | 보존파일 |
| | *.Z/*.gz | 압축파일 |
| | *.tar.gz/*.tgz | 압축한 tar파일 |

### 1.2.4. Linux환경에서 장치의 사용

장치는 콤퓨터를 이루고있는 CPU를 제외한 모든 하드웨어를 말한다. 보통 장치는 자료기억기와 자료완충장치, 장치조종기 등을 의미한다.

Linux환경에서 장치파일은 모두 /dev등록부에 있다. 아래에서 일반장치파일의 형태에 대하여 설명한다.

표 1-4. 장치파일의 형태

| 장치파일 | 설 명 |
|---|---|
| /dev/hd* | IDE하드디스크장치이다. 실례로 hda1은 첫번째 IDE하드디스크의 첫번째 구획을 가리키며 hdb2은 두번째 IDE하드디스크의 두번째구획을 가리킨다. |

표계속

| 장치파일 | 설 명 |
|---|---|
| /dev/sd* | SCSI하드디스크장치를 가리킨다. 실례로 sda1은 첫번째 SCSI하드디스크의 첫번째구획을 가리킨다. |
| /dev/lp* | 병렬입출구장치를 가리킨다. 실례로 lp0은 첫번째 병렬입출구장치를 가리킨다. |
| /dev/cua* | 직렬입출구장치 |
| /dev/tty* | 말단장치 |
| /dev/console | 체계조종탁 |
| /dev/scd* | SCSI CD-ROM장치 |
| /dev/ippp* | PPP장치 |
| /dev/isdn* | ISDN장치 |

/dev등록부안에는 많은 련결파일들이 있는데 이것들을 리용하면 체계에서 장치를 사용하는데 편리하다. 실례로 /dev/cdrom이나 /dev/hdc를 리용하여 cd구동기를 쓸 수 있다.

이밖에도 Linux는 다음과 같은 몇가지 특수한 장치를 가지고있다.

/dev/null 빈장치
/dev/zero 령장치
/dev/loop0 순환장치
/dev/ram0 가상디스크

## 1.2.5. 통용기호

통용기호는 사용자가 등록부나 파일을 쉽게 표현하는데 리용된다. 통용기호를 메타문자라고도 한다.

표 1-5에 일반적인 통용기호와 그에 대한 설명을 주었다.

표 1-5. 일반적인 통용기호

| 통용기호 | 설 명 |
|---|---|
| * | 어떤 문자렬이나 어떤 수에 대응한다. |
| ? | 한개의 문자에 대응한다. |
| [...] | 괄호안의 하나의 문자에 대응한다. |
| \ | 뒤에 오는 문자를 문자자체로 인식한다. |
| & | 지령을 배경에서 처리한다. |
| ; | 지령들을 구분한다. |
| $ | 변수들을 치환한다. |

표계속

| 통용기호 | 설　　명 |
|---|---|
| [abc] | 묶음의 문자들가운데서 한개 문자에 대응된다. 례를 들면 a, b 혹은 c |
| [!abc] | 묶음의 문자들을 제외한 나머지 문자에 대응된다. |
| (cmds) | 자식쉘에서 지령들을 실행한다. |
| {cmds} | 현재쉘에서 지령들을 실행한다. |

통용기호를 사용하면 여러가지 형태의 파일들을 현시하는데 매우 편리하다. 아래에 그 실례를 주었다.

표 1-6. 통용기호를 사용한 파일현시지령

| 지　　령 | 설　　명 |
|---|---|
| ls *.c | 현재등록부안의 모든 c언어원천파일을 현시한다. |
| ls /home/*/*.c | home등록부안의 부분등록부에 있는 모든 c언어원천파일을 현시한다. |
| ls n*.conf | 현재등록부에서 문자 n으로 시작하는 모든 conf파일을 현시한다. |
| ls test?.dat | 현재등록부에서 test로 시작하며 다음에 어떤 한개의 문자가 있는 dat파일을 현시한다. |
| ls [abc]* | 현재등록부에서 첫 문자가 a나 b 또는 c로 시작하는 모든 파일을 현시한다. |
| ls [!abc]* | 현재등록부에서 첫 문자가 a나 b 혹은 c로 시작하지 않는 모든 파일을 현시한다. |
| ls [a-zA-Z]* | 현재등록부에서 첫 문자가 영어자모로 시작하는 모든 파일을 현시한다. |

## 제3절. 쉘지령의 도움말을 얻는 방법

다른 프로그람에서와 마찬가지로 쉘도 직결가능한 도움말(Online Help)기능을 가지고 있다. 그것이 바로 man지령과 info지령이다. man이 처음부터 사용해온 고전적인것이라면 info는 man이후에 만든 새로운것으로서 두 지령의 기능은 기본적으로는 서로 같고 차이점이라면 하나는 인쇄가능한 서식이고 다른 하나는 현시가능한 서식이라는것이다.

### 1.3.1. man을 사용한 도움말 얻기

man지령은 UNIX개발초기부터 존재하였다. man으로 찾을수 있는 정보는 대단히 많다. 이 정보문서는 보통 환경변수 MANPATH가 지정하는 몇개의 등록부에 있다. 문서파일들의 이름이 지령이름과 같으므로 쉽게 해당문서를 찾을수 있다. 실례로 kill지령에 대하여 보기 위해서는 다음과 같은 지령을 주면 된다.

man kill

man에는 몇개의 단순문자지령이 있는데 어떤 자료가 필요할 때 H건을 누르면 도움

말이 현시되며 탈퇴하려면 Q건을 누르면 된다.

man을 사용하기 위해서 MANPATH를 설정해줄 필요는 없다. 실제로 아무것도 설정하지 않아도 필요한 도움말을 볼수 있다. /etc/man.conf에는 기정으로 경로이름이 들어있다. man을 리용할 때 여러개의 추가선택들을 조합하여 쓸수 있다. 만일 파일의 정확한 이름을 모르면 파일이름의 일부와 함께 –k추가선택을 리용한다.

man -k rcv

이 지령은 문자렬 rcv가 포함된 파일이름과 한 행짜리 설명문을 목록화하여 간단하게 보여준다.

man에 대한 보다 상세한 정보를 얻자면 다음과 같이 입력해야 한다.

man man

### 1.3.2. info지령을 사용한 도움말얻기

이 지령은 man보다 새로운 지령으로서 한 페지로부터 다른 페지에로의 련결처리와 목록차림표처리 등 많은 처리를 할수 있다.

가장 좋은 학습방법은 개별교수프로그람(tutorial)을 리용하는것이다. 이것을 실행하기 위해서 다음과 같은 지령을 준다.

info info

만일 info편의프로그람으로 사용자가 요구하는 지령이름을 찾지 못하는 경우 man을 호출하면 된다. 쉘내부지령을 찾으려면 help지령을 사용해도 된다.

man지령과 info지령을 사용하여 매 지령의 도움말을 볼수 있다.

실례로 아래와 같은 지령을 입력하면 그림 1-4와 같은 화면이 표시된다.

$info ls

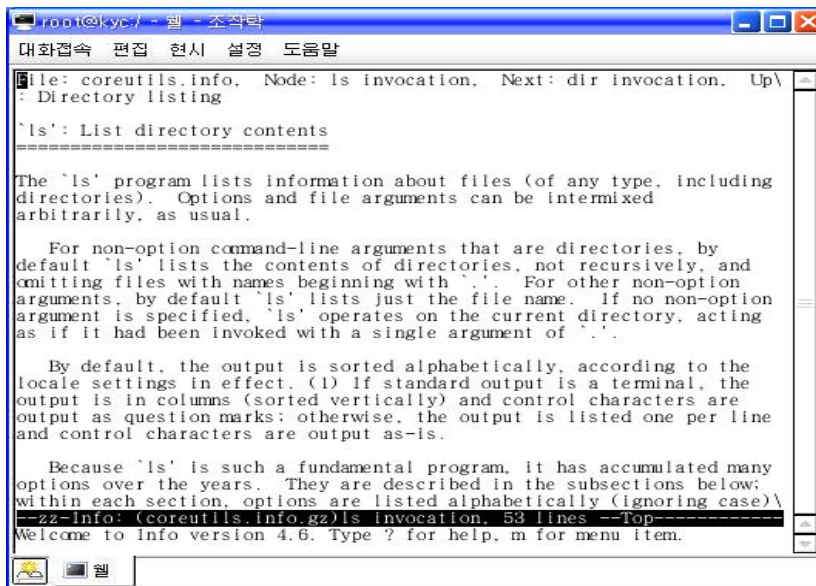



그림 1-4. info지령을 사용한 지령도움말 얻기

사용자는 PageUp건과 PageDown건을 리용하여 다음페지와 전페지를 볼수 있으며 Q건을 누르면 탈퇴한다. 그밖에 사용자는 Ctrl+H건을 눌러 info지령의 도움말을 보고 보다 구체적인 사용방법을 학습할수 있다.

# 제2장. Linux지령과 쉘지령

## 제1절. 파일과 등록부 조작지령

### 2.1.1 파일과 등록부 조작지령

표 2-1. 파일과 등록부 조작지령

| 지 령 | 기 능 |
|---|---|
| ls | 파일과 등록부목록을 표시한다. |
| touch | 빈 파일을 만들고 파일의 변경시간을 기록한다. |
| cp | 파일이나 등록부를 복사한다. |
| mv | 파일이나 등록부를 이동하든가 혹은 그의 이름을 바꾼다. |
| rm | 파일과 등록부를 삭제한다. |
| cat, tac | 본문파일의 내용을 현시한다. |
| more, less | 본문파일의 내용을 페지별로 나누어서 현시한다. |
| head, tail | 본문파일의 앞 혹은 뒤의 일부 행을 현시한다. |
| wc | 지정한 본문파일의 행수, 문자수, 단어수를 통계한다. |
| ln | 파일련결을 만든다. |
| whereis | 체계파일이 있는 경로를 탐색한다. |
| find | 파일체계에서 지정한 파일을 찾는다. |
| grep | 선택한 본문파일에서 지정한 문자렬을 찾는다. |
| pwd | 현재 작업하는 등록부를 현시한다. |
| cd | 등록부를 바꾼다. |
| mkdir | 등록부를 만든다. |
| rmdir | 빈 등록부를 삭제한다. |
| tree | 등록부의 뿌리구조를 현시한다. |

### 2.1.2. 파일과 등록부조작실례

다음의 표에서 파일과 등록부조작지령의 사용실례를 보여준다.

표 2-2. 파일과 등록부조작지령의 사용실례

| 지 령 | 설 명 |
|---|---|
| ls -a | 숨은 파일을 포함하여 현재등록부의 파일과 등록부목록을 표시한다. |
| ls -l | 파일과 등록부에 대한 전체경로를 표시한다. |
| pwd | 현재 작업하는 등록부를 현시한다. |
| mkdir /home/kim | 절대경로로서 등록부를 만든다. |
| mkdir -p mydoc/FAQ | 상대경로로서 등록부를 만든다. |
| touch abc bcd | 두개의 0byte파일을 만든다. |
| cd mybin | mybin등록부로 들어간다. |
| cd - | cd지령전의 등록부로 들어간다. |
| cd .. | 현재등록부의 웃준위등록부로 들어간다. |
| tree | 현재등록부의 아래준위등록부구조를 현시한다. |
| cp /bin/?sh | 기호 ?를 사용하여 여러개의 파일을 현재등록부에 복사한다. |
| cp /bin/cpio mybin | 한개의 파일 /bin/cpio를 mybin등록부에 복사한다. |
| cp abc bcd mydoc | 두개의 파일 abc와 bcd를 mydoc등록부에 복사한다. |
| cp /usr/bin/[yz]* | [ ]기호와 *기호를 사용하여 여러개의 파일을 현재등록부에 복사한다. |
| cp -r /etc/skel | /etc/skel등록부안의 모든 파일을 현재의 등록부에 복사한다. |
| mv FAQ bash-FAQ | FAQ를 bash-FAQ로 이름을 바꾼다. |
| mv [yz]* usr/ | [ ]기호와 *기호를 사용하여 여러개의 파일을 usr/등록부아래에 복사한다. |
| rm ash | ash파일을 삭제한다. |
| rm .* | 현재 등록부안의 숨겨진 모든 파일을 삭제한다. |
| rm -r usr/ | usr등록부와 그안의 모든 내용을 삭제한다. 삭제하기전에 물어본다. |
| rm -rf usr/ | 삭제하겠는가를 묻지 않고 usr등록부의 모든 내용을 삭제한다. |
| rmdir skel | 빈 등록부인 skel을 삭제한다. |
| ln cpio edit1 | cpio의 련결파일 edit1을 만든다. |
| ln -s cpio edits1 | cpio의 기호련결파일 edits1을 만든다. |
| cat mylist | 파일 mylist의 내용을 표시한다. |
| more myalllist | 파일 myalllist의 내용을 화면에 나누어서 표시한다. |

표계속

| 지　　령 | 설　　명 |
|---|---|
| less myalllist | 파일 myalllist의 내용을 화면에 나누어서 표시한다. |
| wc myalllist | 지정한 본문파일의 행수, 문자수, 단어수를 통계한다. |
| wc -l myalllist | 지정한 본문파일의 행수를 통계한다. |
| head -4 myalllist | 지정한 본문파일을 앞으로부터 4번째 행까지 현시한다. |
| tail -4 myalllist | 지정한 본문파일을 뒤로부터 4번째 행까지 현시한다. |
| tail +45 myalllist | 지정한 본문파일의 45번째 행부터 마지막행까지 현시한다. |
| tail +15 myalllist \|head -3 | 지정한 본문파일의 15번째행에서 3개행을 현시한다. |
| grep "my" mylist | mylist파일에서 문자렬 my를 찾는다. |
| grep "my*" mylist myalllist | 지정한 여러개의 파일에서 문자렬 my를 찾는다. |
| find -name "my*" | 현재등록부에서 이름이 "my"로 시작하는 파일을 찾는다. |
| find /home -user "kim" | /home등록부에서 사용자의 이름이 kim인 파일을 찾는다. |
| whereis tar | 확장자가 tar인 프로그람의 위치를 찾는다. |

### 2.1.3. 등록부탄창조작

쉘에서 작업할 때 등록부를 우, 아래로 이동시키는 경우가 많다. 이 경우 자주 이동하는 등록부를 등록부탄창에 넣어두면 편리하게 이동할수 있다. pushd내부지령은 탄창에 등록부들을 넣으며 popd지령은 그것들을 없앤다.(실례 2-1) 등록부탄창에서는 왼쪽의 등록부가 제일 최근에 입력된 등록부이다. 내부지령 dirs로 탄창안의 전체 등록부목록을 현시할수 있다.

- dirs내부지령

내부지령 dirs를 -l추가선택과 함께 쓰면 전체 경로이름으로 탄창의 내용을 현시한다. 추가선택을 주지 않으면 dirs는 홈등록부를 현시한다. +n추가선택을 사용하면 dirs는 등록부목록의 왼쪽에서부터 계산하여 n번째 항목을 현시하고 -n추가선택을 사용하면 등록부목록의 오른쪽에서부터 n번째항목을 현시한다.

- pushd와 popd지령

인수로 등록부이름을 주면 pushd지령은 등록부탄창에 새로운 등록부를 추가하는것과 동시에 그 등록부로 이동한다. 인수로 +n(여기서 n은 수값이다.)을 주면 pushd는 탄창의 왼쪽에서부터 n번째 등록부를 제일 왼쪽의 등록부와 교환한다. -n인수도 같은 내용이지만 단지 오른쪽에서부터라는데 차이가 있다. 아무런 인수도 없으면 pushd는 등록부탄창의 제일 우에 있는 2개의 등록부들을 교환한다.

popd지령은 탄창의 제일 우의 등록부를 없애며 해당 등록부로 이동한다. +n추가선택을 지정하면 popd는 탄창의 왼쪽에서부터 n번째 등록부를 삭제한다.

**실례 2-1.**

```
1    $ pwd
     /home/kim
     $ pushd ..
     /home ~
     $ pwd
     /home
2    $ pushd #Swap the top directories on the stack
     ~/home
     $ pwd
     /home/kim
3    $ pushd perlclass
     ~/perlclass ~ /home
4    $ dirs
     ~/perlclass ~/home
5    $ dirs - l
      /home/kim/perlclass /home/kim /home
6    $ popd
     ~/home
     $ pwd
     /home/kim
7    $ popd
     /home
     $ pwd
     /home
8    $ popd
     bash:popd:Directory stack empty.
```

**설명**

1 첫번째 pwd지령은 현재 작업등록부 /home/kim을 현시한다. 인수로 ..을 주면 pushd는 현재등록부의 부모등록부 /home을 등록부탄창에 넣는다. 새로운 등록부는 두번째 pwd지령에 의해 현시된다. pushd지령의 출력은 등록부환경의 제일 우인 /home을 가리킨다. 등록부탄창의 바닥에는 사용자의 홈등록부인 /home/kim이 보관되여있다.

2 인수가 없는 pushd는 등록부탄창의 제일 우에 있는 2개 등록부들의 위치를 서로 바꾼다. 즉 /home과 /home/kim의 위치가 바뀌여 /home/kim이 등록부탄창의 제일우에 놓이며 사용자의 홈등록부인 /home/kim에로 이동한다.

3 pushd지령은 인수 ~/perlclass를 등록부탄창안에 넣으며 이 등록부로 이동한다.

4 dirs내부지령은 등록부탄창의 내용을 현시한다. ~는 사용자의 홈등록부를 의미한다.

5 -l추가선택을 주면 dirs지령은 ~확장기호대신에 전체경로이름으로 등록부탄창의 내용을 현시한다.
6 popd지령은 등록부탄창의 제일 우에서부터 한개 등록부 /home/kim/perlclass를 삭제한다. 그다음 /home/kim등록부로 이동한다.
7 popd지령은 등록부탄창의 제일 우에 있는 등록부 /home/kim을 삭제한 다음 /home등록부로 이동한다.
8 popd지령은 등록부탄창이 비였기때문에 아무런 등록부도 지우지 못하고 bash:popd:directory stack empty.와 같은 등록부탄창이 비였다는 통보문을 현시한다.

## 제2절. 정보현시지령

### 2.2.1. 정보현시지령

다음의 표에 자주 쓰이는 정보현시지령들을 보여주었다.

표 2-3. 정보현시지령

| 지 령 | 기 능 |
|---|---|
| stat | 지정한 파일의 관련정보를 현시한다. |
| who, w | 현재 등록된 사용자를 현시한다. |
| whoami | 현재 등록가입한 사용자를 현시한다. |
| hostname | 주콤퓨터의 이름을 현시한다. |
| uname | 조작체계의 정보를 현시한다. |
| dmesg | 체계기동정보를 현시한다. |
| top | 현재체계에서 자원소비량이 제일 많은 프로쎄스를 표시한다. |
| du | 지정한 파일이 하드디스크에서 사용하는 공간의 총용량을 현시한다. |
| df | 파일체계가 하드디스크공간을 차지한 상태를 표시한다. |
| free | 현재상태의 기억기와 교환구역의 사용량을 표시한다. |
| ifconfig | 망대면부정보를 현시한다. |
| ping | 망접속상태를 표시한다. |
| netstat | 망상태정보를 표시한다. |
| locale | 현재언어환경을 표시한다. |
| id | 현재사용자의 ID정보를 표시한다. |

### 2.2.2. 정보현시조작지령의 실례

표 2-4. 정보현시조작지령의 실례

| 지　　령 | 설　　　　명 |
|---|---|
| stat abc | 파일 abc의 련관정보를 현시한다. |
| uname -r | 조작체계의 판본정보를 현시한다. |
| ifconfig eth0 | 망대면부 eth0의 정보를 현시한다. |
| ping 192.168.1.100 | 자기콤퓨터와 주콤퓨터 192.168.1.100과의 망련결성을 검사한다. |
| ping www.sin.com.kp | 자기콤퓨터와 주콤퓨터 www.sin.com.kp와의 망련결성을 검사한다. |
| netstat -nr | 경로표의 정보를 표시한다. |

## 제3절. 파일압축지령

### 2.3.1. 파일압축지령

다음의 표에 자주 쓰이는 파일압축지령을 보여주었다.

표 2-5. 파 일 압 축 지 령

| 지　령 | 기　　　　능 |
|---|---|
| tar | 여러개의 파일이나 등록부를 하나의 파일로 묶는다. |
| gzip | 파일과 등록부의 압축(풀기)이며 파일을 압축한 후에 확장자 gz를 붙인다. |
| compress | 파일과 등록부의 압축(풀기)이며 파일을 압축한 후에 확장자 Z를 붙인다. |
| bzip2 | 파일과 등록부의 압축(풀기)이며 파일을 압축한 후에 확장자 bz2를 붙인다. |
| zcat | 압축된 본문파일의 내용을 현시한다. |

### 2.3.2. 파일압축지령조작의 실례

표 2-6에 일반적인 파일압축지령의 조작실례를 보여주었다. Linux환경에서는 보통 tar지령을 써서 매개 파일들을 압축하여 묶거나 압축을 해제하기도 한다.

표 2-6. 파일압축지령과 그 실례

| 지　　령 | 설　　명 |
|---|---|
| tar -zcvf myball.tar.gz mydir | mydir등록부를 myball.tar.gz라는 패키지로 묶고 gzip로 압축한다. |

표계속

| 지　　령 | 설　　명 |
|---|---|
| tar -zxvf myball.tar.gz | 패키지 myball.tar.gz의 압축을 gzip로 해제한다. |
| tar -zcvf myball.tar.Z mydir | mydir등록부를 myball.tar.Z라는 패키지로 묶고 compress로 압축한다. |
| tar -zxvf myball.tar.Z | 패키지 myball.tar.Z의 압축을 compress로 해제한다. |
| tar -jcvf myball.tar.bz2 mydir | mydir등록부를 myball.tar.bz2라는 패키지로 묶고 bzip2로 압축한다. |
| tar -jxvf myball.tar.bz2 | 패키지 myball.tar.bz2의 압축을 bzip2로 해제한다. |

## 제4절. 체계관리지령

### 2.4.1. 일반적인 체계관리지령

표 2-7. 일반적인 체계관리지령

| 지　　령 | 기　　능 |
|---|---|
| mkbootdisk | 체계기동디스크만들기 |
| mount | 파일체계탑재 |
| umount | 파일체계해제 |
| useradd | 사용자추가 |
| passwd | 사용자의 통과어를 바꾸기 |
| su | 지정한 사용자로 절환 |
| chmod | 파일 혹은 등록부의 호출권한을 바꾸기 |
| chown | 파일 혹은 등록부의 사용자를 바꾸기 |
| ps | 프로쎄스검사 |
| kill | 지정한 프로쎄스끄기 |
| rpm | Red Hat Linux의 프로그람패키지관리 |

### 2.4.2. 체계관리지령과 그 조작실례

표 2-8에 자주 쓰이는 일부체계관리지령의 조작실례를 보여주었다.

표 2-8. 체계관리지령의 실례

| 지 령 | 설 명 |
|---|---|
| mkbootdisk 'uname -r' | 체계기동디스크를 만든다. |
| mount | 체계에 자동으로 탑재되여있는 장치를 알수 있다. |
| mount /mnt/cdrom | cd구동기를 탑재한다. |
| umount /mnt/cdrom | cd구동기의 탑재를 해제한다. |
| mount /mnt/floppy | 플로피구동기를 탑재한다. |
| umount /mnt/floppy | 플로피구동기의 탑재를 해제한다. |
| mount -t vfat /dev/hda10 /mnt/win | FAT체계를 /mnt/win등록부에 탑재시킨다. |
| umount /mnt/win | /mnt/win등록부의 탑재를 해제한다. |
| useradd user01 | 새로운 사용자 user01을 추가한다. |
| passwd user01 | 사용자 user01의 통과어를 만든다. |
| su - | root사용자로 절환한다. |
| chmod 700 mydoc | mydoc에 권한 700을 부여한다. |
| chmod 770 staff | staff에 권한 770을 부여한다. |
| chmod o-r,o-x staffdoc | staffdoc등록부에 대한 다른 사람들의 읽기 및 쓰기권한을 취소한다. |
| chmod g-w placard | placard에 대하여 그룹의 쓰기권한을 취소한다. |
| chmod +x watch-stat | 스크립트 watch-stat에 실행권한을 부여한다. |
| chown staff staff-doc | staff-doc의 파일소유자를 staff로 한다. |
| ps | 현재 등록가입한 사용자의 프로쎄스상태를 보여준다. |
| ps au | 모든 사용자들의 프로쎄스를 보여준다. |
| ps lew | 모든 프로쎄스의 정보를 구체적으로 현시한다. |
| ps auxw | 현재조종말단에 없는 프로쎄스도 모두 포함하여 표시한다. |
| kill 2277 | 번호가 2277인 프로쎄스를 완료시킨다. |
| kill -9 2277 | 번호가 2277인 프로쎄스를 강제로 완료시킨다. |
| killall -9 cat | 이름이 cat인 모든 프로쎄스를 완료시킨다. |
| rpm -qa\|grep webmin | 현재 체계에 webmin이라는 rpm이 설치되여있는지 조사한다. |

표계속

| 지　　　　　령 | 설　　　　　명 |
|---|---|
| rpm -qip webmin-0.980-1.noarch.rpm | 패키지에 대한 정보를 현시한다. |
| rpm -qlp webmin-0.980-1.noarch.rpm | 이 rpm파일이 체계에 어떤 파일들로 설치되는지 현시한다. |
| rpm -ivh webmin-0.980-1.noarch.rpm | webmin의 rpm패키지를 설치한다. |
| rpm -Uvh webmin-0.980-1.noarch.rpm | webmin의 rpm패키지를 갱신한다. |
| rpm -q webmin | 이미 webmin패키지가 설치되여있는가를 조사한다. |
| rpm -V webmin | webmin패키지를 시험설치해본다. |
| rpm -e webmin | 이미 설치된 webmin패키지를 삭제한다. |
| rpm -qf /usr/sbin/logrotate | 파일 /usr/sbin/logrotate가 어느 rpm에 의해 설치되는지 검사한다. |

### 2.4.3. 체계의 끄기와 재기동

체계의 끄기와 재기동은 실행준위(runlevel)의 절환에 의해서 진행된다.
실례로 다음의 지령을 사용하면 콤퓨터를 끌수 있다.
#init 0
또한 체계를 재기동하기 위해서는 아래의 지령을 사용한다.
#init 6
그밖에도 halt지령과 reboot지령을 사용하여 콤퓨터를 끄거나 재기동할수 있다.

지령 init는 즉시에 콤퓨터를 끄거나 재기동하게 하므로 많은 사용자가 체계를 사용할 경우에는 다른 사용자에게 경고문을 보내여 매 사용자가 자기의 작업을 끝내도록 하려면 shutdown지령을 사용하여야 한다. 실례로 아래의 지령은 모든 사용자에게 5분후에 체계가 꺼진다는것을 알리고 콤퓨터를 끈다.

```
#shutdown -h +5\
"System will be down in 5 minutes, Please save your work."
```

다음의 표에 init지령에서 쓸수 있는 실행준위번호를 보여주었다.

표 2-9. init지령에서 쓸수 있는 실행준위번호

| 실행준위번호 | 내　　　　　용 |
|---|---|
| 0 | 체계끝내기 |
| 1 | 단일사용자가 사용하는데 적합하도록 체계가 최소화된 상태로 기동한다. |
| 2 | 다중사용자방식이나 NFS는 지원하지 않는다. |
| 3 | 다중사용자방식으로 망련결을 가능하게 한다. |

표계속

| 실행준위번호 | 내　　　　용 |
|---|---|
| 4 | 사용하지 않음. 사용자정의준위로 사용될수 있다. |
| 5 | X Windows로 기동할 때 사용한다. |
| 6 | 재기동 |

## 제5절. 작업효률을 높이는 몇가지 방법

### 2.5.1. 자동지령보충

보통 bash에서 지령을 입력할 때 지령을 완전히 입력하지 못해도 bash는 사용자가 입력한 지령을 판단한다. 실례로 현재작업등록부에 다음의 부분등록부와 파일이 있다고 하자.

```
$ls
system/  myprogram
```

이때 system부분등록부로 들어가려면 다음의 지령을 주어야 한다.

$cd system

그러나 bash는 이 작업을 완성할수 있는 다른 방법을 제공하고있다. system은 현재등록부안의 s로 시작하는 부분등록부이므로 bash는 s를 입력한 후에 사용자가 어떤 일을 하려고 하는가를 판단한다.

$cd s

우의 지령과 같이 문자 s를 입력한 다음 system을 입력하기 위해서 tab건을 누른다.

$cd s<tab>

tab건을 누르면 bash는 화면에 이 지령을 보충하여 현시한다. 보충된 지령이 요구한 지령이 아닐 때 다시 tab건을 누른다. 정확한 지령이 입력되였다면 enter건을 눌러 실행한다.

마찬가지로 다음과 같은 조작으로 myprogram파일을 실행할수 있다.

$./m<tab>

입력할 지령이 비교적 짧을 때에는 이 조작의 편리성을 별로 느끼지 못하지만 지령이 길 때에는 그 우점을 알수 있다. 현재등록부에 s로 시작하는 등록부가 여러개 있을 때에는 서로 다른 파일로 인정될수 있는 문자까지 입력한 후에 <tab>건을 누른다.

$ls

system/ sybase/  myprogram

현재 등록부안에 s로 시작하는 2개의 등록부가 있는데 사용자가 system등록부로 들어가려고 $cd s<tab> 를 입력하면 주어진 정보가 부족하기때문에 bash는 사용자가 입력하려는 등록부를 알지 못한다. 이때 bash는 경고음을 울리여 사용자가 부족한 정보를 보충하도록 한다. 경고음이 울린 후에 bash는 입력지령을 변경시키지 않으므로 사용

자는 원래의 지령에 기초하여 ys를 추가적으로 입력한 다음 <tab>건을 누르면 된다.

$cd sys<tab>

이때 bash는 충분한 정보로써 지령 cd system을 완성하게 된다.

사용자가 지령을 입력할 때 <tab>건을 누르면 bash는 지령을 보충하려고 시도한다. 다시 <tab>건을 누르면 체계는 사용자가 입력할수 있는 지령들의 목록을 현시한다. 다시말하여 <tab>건을 누른 후에 경고음이 울렸을 때 다시 <tab>건을 누르면 이때 bash는 조건에 부합되는 모든 등록부와 파일을 현시하여 사용자가 계속하여 보다 많은 정보를 입력할수 있게 한다. 실례로

$Red Hat-<tab><tab>

라고 지령을 입력하면 그 결과는 다음과 같다.

Red Hat-cdinstall-helper
Red Hat-config-date
Red Hat-config-kickstart
Red Hat-config-language
Red Hat-config-mouse
Red Hat-config-network
Red Hat-config-network-cmd

### 2.5.2. 지령리력

bash는 사용자가 반복작업을 피하도록 하기 위해 여러측면에서 편리성을 제공한다. 앞에서 소개한 자동지령보충과 이제 소개하려는 지령리력과 별명이 그 대표적실례이다.

여기서는 bash의 리력기능에 대하여 서술한다.

**1. 리력**

리력기능으로 지령행에 입력한 지령들을 지정한 개수만큼 보관할수 있다. 등록가입한 후 입력한 지령들은 기억기의 리력목록안에 기억된다. 등록탈퇴하면 이것들이 리력파일에 덧붙게 된다. 사용자는 지령을 다시 입력하지 않아도 리력목록으로부터 지령들을 다시 호출할수도 있고 실행할수도 있다.

내부지령 history는 리력목록을 현시한다. 리력파일의 기정이름은 .bash_history이며 홈등록부에 있다. bash가 리력파일을 호출할 때 HISTORY파일의 변수 histsize에서 지정한 개수의 지령을 쉘기억기에 복사한다. 기정크기는 500이다. HISTFILE변수는 지령이 기억되는 지령리력파일(기정으로는 .bash_history)의 이름을 규정한다. 설정되여있지 않으면 쉘이 탈퇴할 때 리력목록을 파일에 보관하지 않는다.

리력파일은 등록탈퇴할 때마다 계속 증가한다. HISTFILESIZE변수는 리력파일안에 넣을수 있는 최대행수를 조종한다. 기정값은 500이다. fc -1지령을 사용하면 리력파일내용을 현시할수도 있고 편집할수도 있다.

표 2-10. 리 력 변 수

| 변 수 이 름 | 설 명 |
|---|---|
| FCEDIT | fc지령을 사용하는 편집기의 경로이름이다. |
| HISTCMD | 리력목록안에서 현재지령의 리력번호 혹은 색인번호이다. HISTCMD의 설정을 해제하거나 다시 설정하면 자기의 특수한 속성은 사라진다. |
| HISTCONTROL | ignorespace값으로 대입되면 공백문자로 시작되는 행들은 리력목록에 보관되지 않는다. 만약 ignoredups값을 대입하면 리력목록과 일치하는 행은 보관되지 않는다. ignoreboth값은 이 두가지 추가선택을 합친것이다. HISTCONTROL의 값이 설정해제되거나 우의 값이 아닌 다른 값으로 설정하면 구문해석기가 읽은 모든 행들이 리력목록에 기억된다. |
| HISTFILE | 지령목록을 보관하는 파일을 규정한다. 기정은 ~/.bash_history이다. 대입되여있지 않으면 지령리력은 대화형쉘이 탈퇴될 때 보관되지 않는다. |
| HISTFILESIZE | 최대행수는 리력파일안에 규정되여있다. 이 변수에 값을 주면 리력파일의 크기는 제한된다. 이 변수에 지정된 값이상의 행들은 보관되지 않는다. 기정값은 500이다. |
| HISTIGNORE | 리력목록에 보관되여야 할 지령행을 결정하는데 사용되는 패턴목록이다. 두점으로 구분된 패턴은 행의 시작표시가 있고 일치하는 문자를 찾는 일반적인 쉘패턴으로 이루어져있다. hystory지령이 중복패턴을 무시하도록 &를 사용할수 있다. 실례로 ty??:&패턴은 ty다음에 두문자가 있는 지령과 같다. 이 지령들은 리력목록에 보관되지 않는다. |
| HISTSIZE | 지령리력안에 보관되는(즉 기억기에 보관되는) 지령의 개수이다. 기정값은 500이다. |

2. 리력파일로부터의 지령호출

- 방향건

리력파일로부터 지령을 호출할 때 방향건을 리용하여 해당한 지령을 선택할수 있다.(표 2-11) 또한 Delete, Insert, Backspace 등의 표준건을 리용하여 선택한 리력지령을 편집할수 있다. 알맞게 편집한 다음 Enter건을 누르면 지령은 다시 실행된다.

표 2-11. 방 향 건

| | |
|---|---|
| ↑ | 리력목록의 웃방향으로 이동한다. |
| ↓ | 리력목록의 아래방향으로 이동한다. |
| → | history지령의 오른쪽으로 유표를 이동한다. |
| ← | history지령의 왼쪽으로 유표를 이동한다. |

- history내부지령
history내부지령은 리력목록에 있는 지령들에 번호를 붙여 현시한다.

**실례 2-2.**

```
$ history
1  ls
2  for i in 1 2 3
3    do
4    echo $i
5  done
6  echo $i
7  man xterm
8  ./configure --with-apache=../apache-1.3.22
9  id -gn
10 id -un
11 id -u
12 ./configure
13 ./configure
14 ./configure i686-Linux
15 ./configure --help
16 man baswh
17 man bash
18 history
19 history
```

**설명**

history내부지령은 리력목록으로부터 번호가 붙은 지령들의 목록을 현시한다.

- **!를 리용한 지령고속실행**
!를 써서 이미 실행했던 지령을 고속으로 실행할수 있다.
지령형식은 다음과 같다.

```
$ !<지령사건번호>
$ !<지령의 앞문자들>
```

여기서 <지령사건번호>는 history지령에 의하여 현시되는 리력목록에서 제일 앞에 있는 지령의 번호이다. 그리고 <지령의 앞문자들>은 리력목록의 지령에서 앞부분의 일부문자들이다. 우의 실례에서 ls지령을 실행시키기 위해서는 다음과 같이 입력한다.

$ !1

현재 실행하려는 지령이 ./configure지령이라면 다음과 같이 지령을 입력하면 된다.

# !./con

즉 방향건을 많이 누르지 않고도 ./configure지령을 실행할수 있다.

또한 이전에 kate /etc/httpd/conf/httpd.conf지령을 실행했다면 다음과 같은 지령을 주어 실행할수 있다.

# !kate

물론 # !ka라고 해도 실행된다.

리력은 grep와 조합되여 자주 사용한다.

[root@localhost /root]# history | grep configure

8 ./configure --with-apache=../apache-1.3.22

13 ./configure

14 ./configure i686-Linux

15 ./configure --help

[root@localhost /root]# !8

./configure --with-apache=../apache-1.3.22

그리고 지령번호에 -부호를 붙여서 실행시킬수도 있다.

실례로 [root@localhost /root]# !-1을 실행하면 리력목록에서 제일마지막 지령이 실행된다.

## 2.5.3. 지령별명

- alias와 unalias지령

별명은 어떤 지령을 사용자가 알맞게 줄여서 재정의할수 있도록 bash쉘에서 지원하는 기능이다. 별명은 어떤 지령이 많은 추가선택이나 인수, 기억하기가 시끄러운 문법을 가지고있을 때 사용하면 효과적이다. 지령행에서 정의한 별명을 자식쉘들에서는 쓸수 없다.

별명은 쉘스크립트안에서도 사용할수 있으나 스크립트안에서 정의하여 사용하지 않는 한 경우에 따라 동작하지 않을수도 있으므로 주의하여야 한다.

별명을 붙이는 지령은 alias이고 별명을 해제하는것은 unalias이다.

문법은 다음과 같다.

```
alias 별명='실지지령'
unalias 별명
```

여기서 《별명》은 만들려는 별명이고 《실지지령》은 쉘의 실지지령이다.

그러면 실례를 들면서 구체적으로 지령을 살펴보자.

- **별명목록제시**

아무런 인수도 없는 alias내부지령은 설정된 모든 별명들의 목록을 현시한다. 먼저 별명이 현시되고 다음에 그것이 의미하는 실제지령이 현시된다.

**실례 2-3.**

```
$ alias
alias co='compress'
alias cp='cp -i'
alias mroe='more'
alias mv='mv -i'
alias ls='ls -colorztty'
alias uc='uncompress'
```

**설명**

alias지령은 매 지령에 해당한 별명을 현시하며 실지지령은 =부호 다음에 현시된다.

**- 별명만들기**

alias지령은 별명을 만드는데 사용된다. 첫번째 인수는 별명이름이다. 행은 별명이 실행될 때 사용되는 지령이나 지령묶음으로 구성된다. Bash쉘에서 별명은 인수들을 가질수 없다. 다중지령은 반두점으로 분리되며 공백들과 메타문자들을 포함하는 지령들은 단일인용부호로 묶어준다.

**실례 2-4.**

```
1  $ alias m=more
2  $ alias mroe=more
3  $ alias lF='ls -alF'
4  $ alias r='fc -s'
```

**설명**

1 more지령의 별명을 m으로 설정한다.
2 more지령의 별명을 mroe로 설정한다.
3 별명의 정의내용이 공백을 포함하므로 단일인용부호안에 넣었다. lF는 지령ls -alF의 별명이다.
4 별명 r는 리력목록으로부터 지령들을 재호출할 때 fc -s대신에 사용된다. 실례로 r vi는 vi문자를 포함하는 리력목록안의 제일 마지막지령을 재실행한다.

**- 별명삭제하기**

unalias지령은 별명을 삭제하기 위해 사용된다. 별명을 일시적으로 비활성화할 때에는 \기호를 주고 별명이름을 주면 된다.

**실례 2-5.**

```
1   $ unalias mroe
2   $ \ls
```

**설명**

1 unalias지령은 정의된 별명목록에서 별명 mroe를 지운다.
2 별명 ls의 사용을 일시적으로 중지한다.

# 제3장. Linux쉘지령들의 간단한 리용

## 제1절. 지령행기초

대화형쉘을 사용할 때 쉘은 입력을 위해 재촉문을 내보낸다. 재촉문이 나타나면 지령을 입력시킬수 있다는것을 의미한다. bash쉘은 4개의 재촉문을 제공하는데 첫번째가 화페기호($)이고 두번째가 크기기호(>)이며 세번째와 네번째는 PS3과 PS4이다. PS3과 PS4는 후에 언급하도록 한다. 재촉문은 쉘이 대화형일 때 현시된다. 이 재촉문은 변경시킬수 있다. 변수 PS1에는 1차재촉문이 기본값 $로 저장되여있다. 이것은 보통 등록가입이 끝나고 사용자의 입력을 기다릴 때 나타난다. 변수 PS2는 2차재촉문인데 >기호가 저장되여있다. 이 2차재촉문은 지령의 일부만을 입력하고 Enter건을 누르면 나타난다. 1차재촉문과 2차재촉문은 변경시킬수 있다.

**- 1차재촉문**

1차재촉문의 기본값은 화페기호($)이다. 보통 재촉문은 /etc/bashrc안에 정의되여있거나 사용자초기파일 .bash_profile 혹은 .profile(Bourne쉘)안에 정의되여있다.

**실례 3-1.**

```
1   $ PS1="$(uname -n) >"
2   chargers >
```

**설명**

1 첫번째 재촉문은 화페기호($)이다. PS1에 사용자이름(uname -n)과 >기호를 지정한다.
2 새로운 재촉문이 현시된다.

**- 2차재촉문**

PS2변수에는 2차재촉문이 대입된다. 그 값은 기정으로 현시장치인 표준오유에 쓰인다. 이 재촉문은 지령이 실행되지 않았거나 지나치게 많은 입력을 주었을 때 나타난다. 기정으로 대입된 2차재촉문기호는 >이다.

## 제2절. 방향바꾸기와 파이프

### 3.2.1. 입출력과 방향바꾸기

쉘이 기동하면 3개의 파일 즉 표준입력, 표준출력 그리고 표준오유 파일을 연다.

일반적으로 표준입력으로는 건반을 사용하며 파일서술자 0을 할당한다. 표준출력으로는 화면을 리용하며 파일서술자 1로 설정된다. 표준오유 역시 화면을 사용하며 파일서술자 2로 설정된다.

하지만 입력을 파일에서 받아야 할 때도 있고 출력이나 오유정보를 파일에 보관해야 하는 경우도 있다. 이러한 작업은 I/O방향바꾸기(redirection)를 사용하여 해결할수 있다. 방향바꾸기연산자목록을 표 3-1에 보여준다.

표 3-1. 방향바꾸기

| 방향바꾸기연산자 | 의미 |
|---|---|
| < | 입력방향바꾸기 |
| > | 출력방향바꾸기 |
| >> | 출력에 첨가 |
| 2> | 오유방향바꾸기 |
| &> | 출력과 오유방향바꾸기 |
| >& | 출력과 오유방향바꾸기 |
| 1>&2 | 출력을 오유로 내보낸다. |
| 2>&1 | 오유를 출력으로 내보낸다. |
| <>파일이름 | 장치파일(/dev로부터)이면 표준입력과 표준출력에 모두 파일을 사용한다. |

다음에 방향바꾸기에 대한 구체적인 실례와 설명을 함께 주었다.

실례 3-2.

```
1   $ tr '[A-Z]' '[a-z]' < myfile # Redirect input
2   $ ls > lsfile                 # Redirect output
    $ cat lsfile
    dir1
    dir2
    file1
    file2
    file3
3   $ date >> lsfile              # Redirect and append output
    $ cat lsfile
    dir1
    dir2
    file1
    file2
    file3
    Sun Dec 2 12:57:22 PDT 2007
4   $ cc prog.c 2> errfile        # Redirect error
5   $ find -name \*.c -print > foundit 2> /dev/null
    # Redirect output to foundit and errors to /dev/null, respectively.
6   $ find -name \*.c -print >& foundit
    # Redirect both output and errors to foundit.
```

```
7   $ find -name \*.c -print > foundit 2 >&1
    # Redirect output to foundit and send errors to where output
    # is going; i.e. foundit
8   $ echo "File needs an argument" 1 >& 2
    # Send standard output to error
```

**설명**

1 표준입력은 Linux의 tr지령의 입력을 건반이 아니라 파일 myfile로부터 실행되도록 방향을 바꾼다. 모든 대문자들을 소문자로 바꾼다.
2 현시장치에 출력을 보내지 않고 파일 lsfile에 출력되도록 방향을 바꾼다.
3 date지령의 출력이 방향바꾸어지고 파일 lsfile에로 첨가된다.
4 C프로그람원천파일 prog.c가 번역된다. 번역이 실패하면 표준오유가 errfile로 방향바꾸기된다.
5 find지령은 *.c로 편성한 파일들을 현재작업등록부에서 검사하기 시작한다. 그 다음 foundit라는 파일에 파일이름들을 써넣는다. find지령의 오유는 /dev/null에 보내진다.
6 find지령은 *.c로 편성한 파일이름들을 현재작업등록부에서 검사하기 시작한다. 그 다음 foundit라는 파일에 파일이름들을 써넣는다. 오유도 역시 foundit파일에 보낸다.
7 6과 같다.
8 echo지령은 표준오유에 대한 통보문을 내보낸다.

### 3.2.2. 파이프(pipe)

**파이프(pipe)란 무엇인가?**

한 프로쎄스의 표준출력과 다른 프로쎄스의 표준입력을 서로 련결시키는 방법이다. 프로쎄스사이의 통신인 IPC(Interprocess communication)중에서 가장 오래된것으로서 최초의 UNIX판본으로부터 지금까지 계속 사용되는 방법이며 특히 쉘에서 많이 사용된다.

파이프를 리용하면 어느 한 지령의 출력을 다른 지령의 입력으로 보낼수 있다. 쉘은 파일서술자를 닫았다가 여는 방법으로 파이프를 진행한다. 하지만 파일서술자를 직접 파일에 지정하지 않고 파이프체계호출로 생성되는 파이프서술자에 파일서술자를 지정한다. 부모쉘이 파이프파일서술자를 작성한 다음 파이프흐름에 있는 매 지령에 대하여 자식프로쎄스를 생성시킨다. 매개 프로쎄스가 파이프서술자를 조종하게 하여 한 프로쎄스는 파이프로 출력하고 다른것은 파이프로부터 입력을 받는다. 간단히 말하여 파이프는 다만 두 프로쎄스가 자료를 공유하는 핵심부완충기라고 할수 있으므로 중간림시파일이 필요없다. 서술자가 설정된 다음 지령은 동시에 실행된다. 한개 지령의 출력이 완충기에 전송되며 완충기가 충만되거나 그 지령이 끝날 때 파이프의 오른쪽지령은 완충기로부터 자료를 읽는다. 핵심부는 한개 프로쎄스가 완충기로부터 읽거나 쓰는동안 다른것이 대기하도록 동기를 맞춘다.

파이프지령의 형식은 다음과 같다.

```
who | wc
```

파이프를 나타내는 기호는 《|》이다.

쉘은 who지령의 출력을 입력으로서 wc지령에 보낸다. 이것은 파이프체계호출에 의해 실현된다. 부모쉘은 2개의 파이프서술자 즉 파이프로부터 읽기와 거기로 써넣기 위한 서술자들을 작성하는 파이프체계호출을 진행한다. 파이프서술자와 련관된 파일들은 림시로 자료를 기억하는데 리용되는 핵심부조종 I/O완충기이므로 림시파일을 만들지 않아도 된다.

부모쉘이 파이프체계호출을 진행하면 두 파일서술자 즉 파이프로부터 읽기 위한 서술자와 파이프에 써넣기 위한 서술자는 되돌려진다. 대입되는 파일서술자들은 파일서술자(file descriptor: fd)표에서 다음번에 리용하여야 할 서술자들인 fd3과 fd4이다.

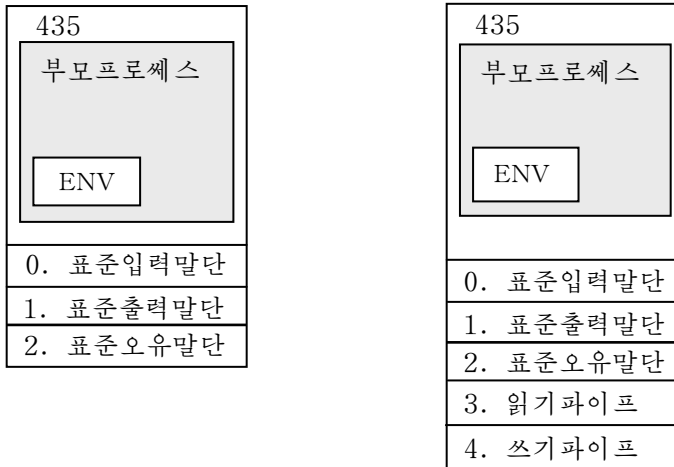



그림 3-1. 파이프를 설정하기 위한 부모프로세스의 파이프체계호출

ls -l /bin | grep sh

-l추가선택을 사용한 ls지령은 /bin등록부에 있는 파일목록과 기타 정보를 보여준다. |기호는 표준출력을 방향바꾸기하여 grep의 입력으로 한다는것을 의미한다. grep지령은 앞지령의 결과에서 sh가 있는 행만을 출력한다. 그 결과는 그림 3-2와 같다.

그림 3-2. ls -l /bin | grep sh지령의 결과

## 제3절. 환경변수

쉘은 자료보관기능을 가지고있다. 보관된 자료는 2개의 문자렬을 가지는데 하나는 변수이고 다른 하나는 그에 대응하는 값이다. 이것을 환경변수라고 한다. 그것은 이 문자렬들이 조작방식을 결정하고 파일의 위치를 지정하여 쉘과 다른 프로그람들에서 쓰이기때문이다.

환경변수는 자신이 생성한 쉘뿐만아니라 그 쉘이 만든 자식쉘이나 프로쎄스에서도 효력이 있다. 환경변수를 국부변수와 구별하기 위해 전역변수라고도 한다. 약속에 따라 환경변수이름은 대문자를 사용한다. 환경변수들은 export내부지령에 의해 반출할수 있는 변수들이다. 변수를 만든 쉘을 부모쉘이라고 한다. 만약 새로운 쉘이 부모쉘로부터 기동되였다면 이것을 자식쉘이라고 부른다.

환경변수는 자기를 만든 쉘로부터 기동한 임의의 자식프로쎄스에서 효력이 있다. 즉 부모프로쎄스에서 만든 환경변수는 자식, 손자프로쎄스에서 다 리용할수 있지만 자식프로쎄스에서 만든 환경변수는 부모프로쎄스에서 리용하지 못하고 오직 자식프로쎄스에서만 효력을 가진다.

몇개의 환경변수들(례: HOME, LOGNAME, PATH, SHELL 등)은 /bin/login 프로그람에 의해 등록가입전에 대입된다. 보통 환경변수들은 사용자의 홈등록부의 bash_profile파일에서 정의한다. 환경변수목록을 표 3-2에 보여주었다.

표 3-2. bash환경변수

| 변수이름 | 의　　미 |
|---|---|
| _(밑선) | 이전 지령의 마지막인수 |
| BASH | bash를 실행하기 위해 사용하는 정확한 경로이름을 준다. |
| BASH_ENV | ENV와 같지만 bash2.0 혹은 그 이상에서만 쓸수 있다. |
| BASH_VERSINFO | 판본 2.0 혹은 그 이상에서 bash의 판본에 대한 통보를 준다. |
| BASH_VERSION | bash의 판본을 준다. |
| CDPATH | cd지령을 위한 탐색경로이다. 지정한 경로가 /, ./, ../으로 시작하지 않으면 쉘은 이 변수값에서 경로를 탐색한다. |
| COLUMNS | 쉘편집방식과 select지령을 위한 편집창문의 너비를 설정한다. |
| DIRSTACK | 등록부탄창의 현재내용이다.(bash판본이 2.0이상일 때) |
| EDITOR | 내부편집기의 경로이름이다.(례: emacs, gmacs 혹은 vi 등) |
| ENV | 스크립트를 포함하여 새로운 bash쉘이 기동될 때 실행되는 환경파일이다. 이 변수에 대입된 파일이름은 보통 .bashrc이다. |
| EUID | 쉘이 기동할 때 현재사용자의 유효 ID를 확장한다. |
| FCEDIT | fc지령을 위한 기정편집기이름이다. |
| FIGNORE | 파일이름을 완성할 때 무시할 뒤붙이를 반두점으로 구분하여 보관한 목록이다. 즉 FIGNORE에 지정된 뒤붙이를 포함하는 파일이름은 파일이름을 완성할 때 제외된다. |

표계속

| 변 수 이 름 | 의 미 |
|---|---|
| FORMAT | 지령파이프행에서 time의 출력을 서식화하기 위해 사용된다. |
| GLOBIGNORE | 파일이름을 확장하는동안 무시되는 파일들의 목록을 나타낸다. |
| GROUPS | 현재 사용자가 소속된 그룹의 배렬을 의미한다. |
| HISTCMD | 현재 지령의 리력목록안에서 리력번호 혹은 색인이다. |
| HISTCONTROL | ignorespace값을 대입하면 공백문자로 시작하는 행들은 리력목록에 입력되지 않는다. ignoredups값을 대입하면 마지막항목과 일치하는 행들은 입력되지 않는다. ignoreboth의 값은 2개의 추가선택을 조합한것이다. 대입하지 않았거나 그이상의 값을 대입하면 해석프로그람에 의해 읽어진 모든 행들은 리력목록우에 기억된다. |
| HISTFILE | 지령리력을 보관할 파일이름이다. 기정으로 ~/.bash_history이다. 이 변수가 설정되여있지 않으면 탈퇴할 때 리력목록을 보관하지 않는다. |
| HISTFILESIZE | 리력파일에 보관할수 있는 최대행수이다. 기정으로 500이다. |
| HISTSIZE | 지령들의 번호가 지령리력안에 들어간다. 기정은 500이다. |
| HOME | 규정된 등록부가 없을 때 cd지령에서 사용하는 홈등록부이다. |
| HOSTFILE | 쉘이 주콤퓨터이름을 완성할 때 사용하는 파일로서 /etc/hosts와 같은 형식으로 파일이름을 보관한다. 다음번에 주콤퓨터이름을 완성하려고 시도하면 새로운 파일의 내용을 자료기지에 추가한다. |
| HOSTTYPE | bash가 실행되고있는 체계의 환경을 자동으로 설정한다. |
| IFS | 내부마당분리기호들인 공백건, 타브, 행바꾸기문자는 지령치환, 순환구조에서의 목록, 입력읽기로부터 마당들을 단어들로 분리하는데 리용한다. |
| INPUTRC | 기정으로 대입된 ~./inputrc를 무시하고 readline기동파일이름을 준다. |
| LANG | LC_로 시작하는 변수가 특별히 대입되여있지 않는 국부목록을 확장하기 위해 사용한다. |
| LC_ALL | LANG값을 제외한 다른 모든 LC_변수이다. |
| LC_COLLATE | 경로이름의 확장과 표현식의 범위, 똑같은 클라스들을 구별할 때와 그리고 경로이름들과 패턴들을 대응시킬 때 대조순서를 결정한다. |
| LC_MESSAGES | $로 처리되는 2중인용부호안의 문자렬을 현시하기 위해 사용하는 국부변수를 결정한다. |
| LINENO | 함수나 혹은 스크립트에서 현재 행번호이다.(1부터 시작된다.) |

표계속

| 변 수 이 름 | 의　　미 |
|---|---|
| MACHTYPE | Bash가 실행되고있는 체계를 서술한다. |
| MAIL | 이 파라메터가 우편파일의 이름으로 설정되고 MAILPATH가 설정되여있지 않으면 쉘은 사용자에게 지정된 파일에 우편이 도착했는가를 알려준다. |
| OLDPWD | 이전 작업등록부이다. |
| OPTRAG | getopts내부지령이 마지막으로 처리한 추가선택의 인수이다. |
| OPTERR | 1로 대입하면 getopts내부지령의 오유통보문을 현시한다. |
| OPTIND | getopts내부지령이 처리할 다음 인수의 색인을 표시한다. |
| OSTYPE | bash를 실행하는 조작체계에 대한 정보를 보관하며 자동으로 설정된다. |
| PATH | 지령들의 검색경로이다. 두점으로 구분된 등록부목록이다. |
| PIPESTATUS | 파이프형의 제일 마지막으로 실행된 전경일감들의 탈퇴상태값을 보관하는 배렬을 의미한다. |
| PPID | 부모프로쎄스의 PID를 의미한다. |
| PROMPT_COMMAND | 이 변수에 지정된 지령은 1차재촉문이 나타나기전에 실행된다. |
| PS1 | 1차재촉문문자렬이다. 기정으로 $이다. |
| PS2 | 2차재촉문문자렬이다. 기정으로 >이다. |
| PS3 | select지령에서 사용되는 선택재촉문문자렬이다. 기정으로 #?이다. |
| PS4 | 추적이 실행될 때 사용되는 오유수정재촉문문자렬이다. 기정으로 +이다. |
| PWD | 현재 작업등록부를 의미한다. |
| RANDOM | 변수를 참고할 때마다 발생하는 란수이다. |
| REPLY | read에 인수를 지정하지 않을 때 설정한다. |
| SECONDS | SECONDS는 쉘이 실행된 이후의 경과시간을 되돌린다. |
| SHELL | 쉘은 기동할 때 이 변수에 지정된 이름에 적합한 쉘환경을 만들어준다. 쉘은 기정값으로 PATH, PS1, PS2, MAILCHECK 그리고 IFS 등을 설정한다. HOME 그리고 MAIL은 login프로그람에 의하여 지정된다. |
| SHELLOPTS | 설정된 쉘추가선택들의 목록을 포함한다. |
| SHLVL | BASH가 기동할 때마다 증가한다. |
| TMOUT | 탈퇴하기전의 입력대기시간을 표시한다. |
| UID | 쉘기동시에 초기화되는 현재사용자 ID를 확장한다. |

### 3.3.1. 환경변수의 설정과 해제

#### 1. 환경변수들의 설정

환경변수를 설정하기 위한 export지령은 새로운 값을 대입하거나 혹은 변수를 대입할 때 사용된다. declare내부지령에 -x추가선택을 주면 같은 동작을 한다.(export지령에 사용되는 변수앞에는 $를 사용하지 않는다.)

**형식**

```
export 변수=값
변수=값; export 변수
declare -x 변수=값
```

**실례 3-3.**

```
export NAME=john
PS1=[\u@\h \W]\$; export PS1
declare -x TERM=sun
```

#### 2. 변수설정해제

읽기전용으로 설정된 변수들을 제외하고 국부변수와 환경변수들은 unset지령을 사용하여 해제할수 있다.

**실례 3-4.**

```
unset name; unset TERM
```

**설명**

unset지령은 쉘기억기에서 변수들을 지운다.

#### 3. 변수값현시

내부지령 echo는 표준출력으로 인수들을 현시한다. echo지령에 -e추가선택을 같이 사용하면 출력표현을 조종할수 있는 ESC코드(확장문자렬)를 사용할수 있다. 표 3-3에 echo추가선택과 ESC코드를 보여주었다 .

표 3-3. echo추가선택과 ESC코드

| 추가선택 | 의 미 |
|---|---|
| -e | ESC코드의 해석을 허용한다. |
| -n | 출력행끝에서의 행바꾸기를 생략한다. |
| -E | ESC문자들의 해석을 무효로 한다.(bash 2.X) |
| ESC코드 | |
| \a | 경고음 |
| \b | 공백건 |
| \c | 마지막에 행바꾸기문자를 사용하지 않는다. |

표계속

| 추가선택 | 의　　미 |
|---|---|
| \f | 용지제공 |
| \n | 행바꾸기 |
| \r | 되돌이 |
| \t | 타브 |
| \v | 수직타브 |
| \\ | 역사선기호(\) |
| \nnn | ASCII코드가 nnn인 문자(8진수) |

ESC코드를 사용할 때 -e추가선택을 반드시 사용해야 한다.

**실례 3-5.**

```
1   $ echo The username is $LOGNAME.
    The username is kim.
2   $ echo -e "\t Hello there\c"
        Hello there$
3   $ echo -n "Hello there"
    Hello there$
```

**설명**

1 echo지령은 현시장치에 인수를 현시한다. 변수치환은 echo지령을 실행하기전에 쉘에 의해 실행된다.

2 echo지령에 -e추가선택을 사용하면 c프로그람작성언어에서 사용하는 확장문자렬을 쓸수 있다. 여기서 $는 쉘재촉문이다.

3 -n추가선택을 대입하면 행바꾸기없이 현시한다.

### 4. printf지령

printf지령은 출력서식을 지정하기 위해 사용한다. C언어의 printf함수와 꼭같은 방식으로 문자렬을 현시한다. 서식지정은 출력을 어떤 형식으로 할것인가를 나타내는 문자렬로 이루어진다. 서식화구조는 %기호다음에 메타문자로 설계된다. 여기서 %f는 류동소수점, %d는 옹근수형10진수를 나타낸다.

지령행재촉문에서 printf --help라고 입력하면 printf에 대한 도움말을 볼수 있다. printf의 판본을 보기 위해서는 printf --version을 입력한다. bash 2.x를 사용하면 내부지령 printf는 /usr/bin에 있는 실행프로그람판본과 같은 형식으로 사용된다.

**형식**

```
printf 서식 [인수…]
```

**실례 3-6.**

```
printf "%10.2f%5d\n" 10.5 25
```

**실례 3-7.**

```
1   $ printf --version
    printf (GNU sh-utils) 1.16
2   $ type printf
    printf is a shell builtin
3   $ printf "The number is %.2f\n" 100
    The number is 100.00
4   $ printf "%-20s%-15s%10.2f\n" "Jody" "Savage" 28
    Jody                Savage               28.00
5   $ printf "|%-20s|%-15s|%10.2f|\n" "Jody" "Savage" 28
    |Jody                |Savage         |     28.00|
6   $ printf "%s's average was %.1f%%. \n" "Jody" $(((80+70+90)/3))
    Jody's average was 80.0%.
```

**설명**

1 printf지령의 GNU판본을 현시한다.

2 bash 2.x를 사용한다면 printf는 내부지령이다.

3 인수 100은 서식화문자렬에서 규정한 %.2f에 의해 10진수의 소수점아래 2자리를 취한 류동소수점형식으로 현시된다. C와는 달리 인수들을 구별하는데 반점이 없다.

4 서식화문자렬은 3개의 조건을 규정한다. 첫번째는 %-20s(20문자 문자렬의 왼쪽 줄맞추기), 다음은 %-15s(15문자 문자렬의 왼쪽 줄맞추기) 그리고 마지막으로 %10.2f(10문자길이의 류동소수점수자를 오른쪽 줄맞추기로 출력시킨다. 여기서 소수부는 2자리이다.)이다. 매 인수는 %부호에 의해 서식화된다. 그러므로 문자렬 jody에는 첫번째 %, 문자렬 savage는 두번째 %, 수값 28은 마지막 %부호에 대응한다.

5 이 행은 4행에 비해 수직기호가 더 첨부되였을뿐이다.

6 printf지령은 문자렬 Jody를 서식화하고 산수연산표현식의 결과를 서식화한다. 2개의 퍼센트기호(%%)는 한개의 퍼센트기호(%)를 현시하기 위해 필요하다.

### 5. 변수확장변경자

변수는 특수한 변경자에 의해 검사되고 변경된다. 변경자는 변수에 값이 있는가를 검사할수도 있고 조건검사의 결과에 따라 새로운 값을 줄수도 있다. 변수확장 변경자에 대한 내용을 표 3-5에 주었다.

표 3-5. 변수변경자

| 변 경 자 | 값 |
|---|---|
| ${변수:-단어} | 변수의 값이 빈값이 아니라면 그 값을 사용한다. 그렇지 않으면 단어로 잠시 치환한다. |
| ${변수:=단어} | 변수의 값이 빈값이 아니라면 그 값을 사용한다. 그렇지 않으면 단어로 영구적으로 치환한다. |

표계속

| 변 경 자 | 값 |
|---|---|
| ${변수:+단어} | 변수의 값이 빈값이 아니라면 변수값을 단어로 잠시 치환한다. 그렇지 않으면 아무것도 치환하지 않는다. |
| ${변수:?단어} | 변수의 값이 빈값이 아니라면 그 값을 사용한다. 그렇지 않으면 단어를 현시하고 쉘로부터 탈퇴한다. 단어가 생략되면 통보문 "parameter null or not set"가 현시된다. |
| ${변수:편차} | 변수문자렬의 부분문자렬을 준다. 편차값은 0에서 시작하여 문자렬의 끝까지이다. |
| ${변수:편차:길이} | 편차에서 시작하여 해당한 길이만큼의 부분문자렬을 변수에서 추출한다. |

두점(:)과 함께 변경자(-, =, +, ?)를 같이 사용하면 변수가 대입되였는가 아니면 빈값인가를 검사할수 있다. 두점이 없으면 빈값으로 대입된 변수도 대입되였다고 본다.

**실례 3-8.**

```
1   $ fruit=peach
2   $ echo ${fruit:-plum}
    peach
3   $ echo ${newfruit:-apple}
    apple
4   $ echo $newfruit
5   $ echo $EDITOR        #More realistic example
6   $ echo ${EDITOR:-/bin/vi}
    /bin/vi
7   $ echo $EDITOR
8   $ name=
    $ echo ${name-Joe}
9   $ echo ${name:-Joe}
    Joe
```

**설명**

1 변수 fruit에 값 peach를 대입한다.

2 특수한 변경자는 변수 fruit에 값이 대입되여있는가를 검사한다. 만약 값이 있으면 peach를 현시하고 없으면 fruit는 plum으로 치환된다.

3 변수 newfruit에는 값이 대입되여있지 않다. 따라서 newfruit는 림시로 값 apple로 치환된다.

4 3행에서의 대입은 다만 림시적이므로 newfruit에는 값이 대입되여있지 않은것으로 된다. 따라서 아무것도 출력되지 않는다.

5 환경변수 EDITOR에는 값이 대입되여있지 않다.
6 :-변경자는 EDITOR를 /bin/vi로 치환한다.
7 EDITOR도 대입되여있지 않다. 아무것도 현시하지 않는다.
8 변수 name은 빈값으로 대입된다. 두점변경자가 없어도 변수는 대입으로 고찰되므로 변수 name에 새로운 값 Joe가 대입된다.
9 두점으로 변수가 설정되여있지 않거나 빈값으로 설정되여있는가를 확인할수 있다. 결과 변수 name은 값 Joe로 치환된다.

**실례 3-9.**

```
1  $ name=
2  $ echo ${name:=Peter}
   Peter
3  $ echo $name
   Peter
4  $ echo ${EDITOR:=/bin/vi}
   /bin/vi
5  $ echo $EDITOR
   /bin/vi
```

**설명**

1 변수 name에 빈값이 대입된다.
2 특수한 변경자 :=는 변수 name에 값이 대입되였는가를 검사한다. 대입되여있으면 변경시키지 않으며 빈값이거나 대입되여있지 않으면 =부호오른쪽의 값을 준다. Peter는 변수가 빈값이므로 name에 대입된다.
3 변수 name은 여전히 값 Peter로 되여있다.
4 변수 EDITOR의 값은 /bin/vi로 대입된다.
5 변수 EDITOR의 값을 현시한다.

**실례 3-10.**

```
1  $ foo=grapes
2  $ echo ${foo:+pears}
   pears
3  $ echo $foo
   grapes
```

**설명**

1 변수 foo는 값 grapes를 가진다.
2 특수한 변경자 :+는 변수가 대입되여 있는가를 검사한다. 만약 대입되여있으면 림시로 foo를 pears로 치환한다. 없으면 빈값이 들어간다.
3 변수 foo는 현재 원래의 값을 가진다.

**실례 3-11.**

```
1   $ echo ${namex:?"namex is undefined"}
    bash: namex: namex is undefind
2   $ echo ${y?}
    bash: y: paremter null or not set
```

**설명**

1 :?변경자는 변수가 대입되여 있는가를 검사한다. 대입되여있지 않으면 ?의 오른쪽 문자렬이 표준오유로 변수이름 다음에 현시된다.
2 통보문이 ?기호다음에 제공되여있지 않으면 쉘은 표준오유로 기정통보문을 현시한다.

**실례 3-12.**

```
1   $ var=notebook
2   $ echo ${var:0:4}
    note
3   $ echo ${var:4:4}
    book
4   $ echo ${var:0:2}
    no
```

**설명**

1 변수에 값 notebook가 대입된다.
2 var의 부분문자렬은 편차 0, 길이 4이므로 n~e의 4개 문자렬로 된다.
3 var의 부분문자렬은 편차 4, 길이 4이므로 b~k의 4개 문자렬로 된다.
4 var의 부분문자렬은 편차 0, 길이 2이므로 n~o의 2개 문자렬로 된다.

**- 부분문자렬의 변수확장**

패턴에 대응되는 인수들은 문자렬의 시작 혹은 끝으로부터 문자렬의 일부를 제거하기 위해 사용한다. 보통 경로의 앞 혹은 뒤부분을 지우기 위해 이 인수들을 사용한다.

표 3-6. 변수확장부분문자렬

| 확 장 | 기 능 |
|---|---|
| ${변수%패턴} | 변수의 뒤부분에서 패턴과 일치하는 제일 작은 부분을 찾아 지운다. |
| ${변수%%패턴} | 변수의 뒤부분에서 패턴과 일치하는 제일 큰 부분을 찾아 지운다. |
| ${변수#패턴} | 변수의 앞부분에서 패턴과 일치하는 제일 작은 부분을 찾아 지운다. |
| ${변수##패턴} | 변수의 앞부분에서 패턴과 일치하는 제일 큰 부분을 찾아 지운다. |
| ${#변수} | 변수에서 문자들의 수를 치환한다. * 혹은 @이면 길이는 위치인수의 개수이다. |

**실례 3-13.**

```
1   $ pathname="/usr/bin/local/bin"
2   $ echo ${pathname%/bin*}
    /usr/bin/local
```

**설명**

1 국부변수 pathname에 /usr/bin/local/bin을 대입한다.
2 %는 패턴 /bin다음에 빈값 혹은 다른 문자들을 포함하는 pathname가운데서 제일 작은 부분을 지운다. 즉 /bin을 해제한다.

**실례 3-14.**

```
1   $ pathname="/usr/bin/local/bin"
2   $ echo ${pathname%%/bin*}
    /usr
```

**설명**

1 국부변수 pathname에 /usr/bin/local/bin을 대입한다.
2 %%는 패턴 /bin다음에 련이어 빈값 혹은 다른 문자들을 포함하는 pathname가운데서 제일 길게 선택되는 부분을 삭제한다. 즉 /bin/local/bin을 해제한다.

**실례 3-15.**

```
1   $ pathname=/home/lilliput/jake/.bashrc
2   $ echo ${pathname#/home}
    /lilliput/jake/.bashrc
```

**설명**

1 국부변수 pathname에 /home/liliput/jake/.bashrc를 대입한다.
2 #는 pathname의 앞부분에서 패턴 /home을 포함하는 제일 작은 부분을 지운다. 즉 경로변수의 시작에 있는 /home이 해제된다.

**실례 3-16.**

```
1   $ pathname=/home/liliput/jake/.bashrc
2   $ echo ${pathname##*/}
    .bashrc
```

**설명**

1 국부변수 pathname에 /home/liliput/jake/.bashrc를 대입한다.
2 ##는 빈값 혹은 다른 문자들이 있으면서 마지막이 /문자로 끝나는 pathname가운데서 제일 길게 선택되는 부분을 삭제한다. 즉 pathname으로부터 /home/liliput/jake를 해제한다.

**실례 3-17.**

```
1   $ name="Ebenezer Scrooge"
2   $ echo ${#name}
    16
```

**설명**

1 변수 name에 문자렬 Ebenezer Scrooge를 대입한다.
2 $(#변수)문법은 변수 name으로 지정된 문자의 개수를 현시한다.
Ebenezer Scrooge는 16개 문자로 구성된다.

### 3.3.2. 일부 환경변수들

1. PATH

환경변수 PATH에는 등록부이름이 차례로 들어있으며 등록부들은 두점으로 구별된다. 다음에 그 실례를 보여주었다.

PATH=/sbin:/bin/:/usr/sbin:/usr/bin:/usr/X11R6/bin

지령행에 지령을 입력하면 이 등록부들에서 이름이 같은 실행가능한 파일이 있는가를 탐색한다. 왼쪽에서 오른쪽으로 가면서 등록부들을 탐색한다. 우의 실례에서는 /sbin에서 시작하여 /usr/X11R6/bin등록부까지 탐색한다. 이미 정의된 등록부목록에 새로운 등록부를 추가할수 있다. 새로운 등록부를 추가하려면 마지막에 다음과 같이 쓰면 된다.

PATH=$PATH:/home/newdir

새로운 등록부를 왼쪽에 추가할수도 있는데 이때는 다음과 같이 하면 된다.

PATH=/home/newdir:$PATH

PATH변수에 현재 작업등록부를 주기 위해서는 간단히 점(.)을 입력하면 된다. 즉 다음과 같이 서술하면 된다.

PATH=/home/newdir:.:$PATH

보안을 위해서는 PATH에 옹근점을 쓰는것을 피해야 한다. 사실 일반사용자가 특권사용자로 되는것은 체계보안에 위험하다. 그런데 일반사용자가 현재 등록부에서 실행권한을 얻어서 트로이목마프로그람을 실행시키면 그 프로그람은 특권사용자의 권한을 얻게 되므로 보안상 문제가 제기될수 있다.

2. PS

환경변수 ps1과 ps2은 쉘이 현시하는 재촉문을 포함한다. ps1은 1차재촉문으로써 사용자가 새로운 지령을 입력할것을 요구한다. ps2은 2차재촉문으로써 한 행보다 더 긴 지령을 입력하였을 때 행이 련속된다는것을 나타낸다.

이 재촉문들은 매우 간단한 문자들로 이루어졌다. 다음의 표에 재촉문을 구성하는데 쓰는 특수문자들을 보여주었다.

표 3-7. 재촉문에 리용되는 문자들

| 조종문자 | 결 과 |
|---|---|
| \t | 현재 시간을 HH:MM:SS형식으로 준다. |
| \d | 날자를《요일 월 일》 형식으로 준다.(즉 Tue May 26) |
| \n | 행바꾸기 |
| \s | 쉘의 이름 |
| \W | 현재 작업등록부 |
| \w | 현재 작업등록부의 전체경로이름 |
| \u | 현재사용자이름 |
| \h | 주콤퓨터이름 |
| \# | 이 지령의 번호 |
| \! | 이 지령의 리력번호 |
| \$ | UID가 0이면 #이고 그외에는 $이다. |
| \nnn | 8진수 nnn으로 응답하는 문자 |
| \\ | \기호 |
| \[ | 문자를 출력할수 없는 코드의 시작으로서 재촉문안에 말단조종문자를 매몰하여 사용할수 있다. |
| \] | 문자를 출력할수 없는 조종코드의 끝 |
| \a | ASCII종울림문자 |
| \@ | 12시간간격으로 AM/PM형식으로 현재시간을 나타낸다. |
| \H | 주콤퓨터이름 |
| \T | 12시간간격으로 HH:MM:SS형식으로 현재시간을 나타낸다. |
| \e | ASCII ESC문자(033) |
| \v | Bash판본인데 례를 들면 2.05이다. |

아래에 재촉문에 대한 실례를 주었다.

**실례 3-18.**

```
1  $ PS1="[\u@\h \w]\\$"
   [kim@homebound kim]$
2  $ PS1="\w:\d>"
   kim:Tue May 18>
3  $ echo "Hello
```

| | |
|---|---|
| 4 | > there"<br>hello<br>there |
| **설명** | |
| 1 | 특수한 ESC코드를 사용하여 원래의 bash재촉문을 대입한다. \u는 사용자의 등록가입이름을 주며 \h는 주콤퓨터이름 그리고 \w는 현재작업등록부이름을 준다. 여기에는 2개의 \기호가 있는데 첫번째 \기호는 두번째 \기호를 처리하는것으로서 결과는 \$로 된다. 화페기호($)는 쉘프로쎄스에서 보호되며 그대로 현시된다. |
| 2 | \w는 현재작업등록부이름, \d로 처리된다. |
| 3 | 2중인용부호는 문자렬 Hello다음에 주어야 한다. |
| 4 | 새로운 행을 입력할 때 2차재촉문이 현시된다. 2중인용부호가 넣어질 때까지 2차재촉문이 현시된다. |

### 3. HISTORY

앞에서 리력에 대하여 약간 설명하였는데 여기서는 좀 더 구체적으로 보기로 하자. 쉘이 대화형방식으로 동작할 때 대부분의 최근지령들은 모두 실행가능하다. 이때 이 지령들은 방향건과 지령번호를 입력하여 실행할수 있다는것을 앞에서 보았다. 만일 어떤 원인에 의해서 지령리력을 삽입할수 없으면 방향건을 리용할수 없다. 이때 지령 fc를 사용하여 가장 최근에 입력된 지령들을 편집하고 수정할수 있다. 지령행에 fc를 입력하면 마치 본문편집기처럼 본문이 출현한다. 이때 그것을 리용하여 정확하게 리력지령들을 만들수 있다. 편집기에서 탈퇴하면 새 지령들이 발생된다.

리력지령을 얻을수 있는 여러가지 방법들이 있다.(표 3-8)

표 3-8. 리력지령편집방법

| 지 령 | 작 용 |
|---|---|
| 방향건 | up건과 down건을 눌러서 지령리력들을 볼수 있다. 매 지령들은 되돌이건에 의하여 실행된다. |
| fc | 기억된 이전 지령들을 본문편집기를 리용하여 수정할수 있다. 편집기에서 탈퇴하면 지령은 실행된다. |
| !string | string으로 시작하는 지령들을 찾아낸다. 실례를 들어 find로 시작하는 이전의 긴 지령을 찾으려면 !find라고 입력하면 된다. 쉘은 최근에 호출한 find지령의 인수들을 찾는다. |
| !?string | string을 포함하고있는 최근 지령들을 찾는다. |
| !-n | 이것은 이전지령의 편의값이다. 만일 n이 1이라면 방금전에 입력했던 지령을 호출한다. |
| !! | !-1과 동일하다. |
| !n | 지령번호가 n인 특수한 지령을 참조한다. |
| ^str1^str2 | 최근의 지령을 반복하는데 str1을 str2로 치환한다. |

## 제4절. 쉘지령호출

bash쉘을 시작하는데는 여러가지 방법이 있으며 인수와 이름을 어떻게 주는가에 따라 그의 동작은 서로 다르다.

### 3.4.1. 등록가입쉘

등록가입쉘은 등록가입과 함께 시작되는 쉘이다. 그 목적은 대화형대화접속을 실현하기 위한것이다. 이것은 사용자쉘이 /etc/passwd에 의하여 시작할 때 체계에서 리용되는 방식이다.

등록가입쉘은 다음의 순서에 따라서 전체 환경변수가 설정되여 초기화된다.

먼저 bash쉘은 여러가지 초기화파일들을 source지령을 사용하여 현재 쉘에 적용한다. 파일을 source지령으로 실행시키면 해당 파일에 설정된 모든 변수들이 현재쉘의 일부가 된다. 즉 자식쉘이 생성되지 않는다. 초기화파일들은 쉘이 등록가입쉘이거나 대화형쉘(등록가입쉘은 아닌) 또는 비대화형쉘(한개의 쉘스크립트)인가에 따라 어떤 파일을 실행하겠는가를 결정한다. 사용자가 등록가입할 때 쉘재촉문이 나타나기전에 체계전체에 적용되는 초기화파일인 etc/profile을 읽어들인다. 그 다음 홈등록부안에 있는 .bash_profile을 실행시킨다. 이것들은 사용자의 별명들과 함수들, 특수한 환경변수들과 기동스크립트들을 대입한다. 사용자의 .bash_profile파일이나 .bash_login파일이 있으면 그 파일을 실행한다.

bash쉘이 초기화파일들을 처리하는 순서를 요약하면 다음과 같다.

만일 /etc/profile이 있으면 실행한다.

다음 ~/.bash_profile이 있으면 실행한다.

그 다음 ~/bashrc이 있으면 실행한다.

그렇지 않고 ~/.bash_login이 있으면 실행한다.

마지막으로 ~/.profile이 있으면 실행한다.

### 3.4.2. 비등록가입쉘

사용자는 지령행에서 직접 새로운 쉘프로그람을 작성할수 있다. 이렇게 작성된 자식쉘은 부모쉘의 설정을 그대로 넘겨받으므로 전체 설정순차를 지킬 필요가 없다. 모든 쉘들은 .bashrc를 실행하면 된다. 이 작용은 -norc추가선택에 의해서만 억제될수 있다.

### 3.4.3. 비대화형쉘

이 형태의 쉘은 하나의 프로쎄스가 다른 프로쎄스를 실행할 때 생긴다. 실례로 지령의 끝에 &기호를 써서 입력하면 비대화형쉘이 시작되여 지령을 실행시킨다.

만일 환경변수 ENV가 설정되여있으면 적당한 환경변수가 설정된 파일이름을 입력해야 한다. 여기서 ENV파일이름은 전체경로의 파일이름이 되여야 한다. 그것은 이때 PATH는 쓰이지 않기때문이다.

### 3.4.4. bourne쉘의 량립성

bash쉘프로그람은 2개의 Linux이름 /bin/bash와 /bin/sh를 가진다. /bin/sh는 UNIX에서의 표준쉘이름이다. 만일 /bin/sh에서 쉘이 시작되면 표준쉘로써 동작한다.

그래서 체계파일은 /etc/profile다음에 .profile만을 실행하며 다른 체계파일들은 쓰이지 않는다.

## 제5절. 일부 특수지령행구조

bash쉘에는 지령행에서 일부규칙이 수정되거나 확장되는 일부 특수한 기능이 있다.

### 3.5.1. 환경변수확장

쉘에서 정의되는 환경변수값은 지령행에서 변수이름앞에 $문자를 대입하여 리용할 수 있다.

다음의 3가지 지령을 보자.

```
echo PWD
echo $PWD
echo ${PWD}
```

첫번째지령은 단순히 3개의 문자 PWD만을 현시한다. 두번째 지령은 현재등록부의 경로이름을 현시한다. 세번째 지령은 2번째 지령과 같은데 다만 대괄호가 덧붙었을뿐이다. 세번째 지령은 환경변수의 이름이 VERSION이고 그것이 파일이름을 구성하는 부분으로 들어간다면 문자렬들이 변수이름끝에서 그대로 출력되는것을 방지하려고할 때 리용한다. 실례로 VERSION번호가 2.1이고 파일의 이름이 code2.1b.c라고 하면 파일의 이름은 다음과 같이 표시할수 있다.

```
ls -l code${VERSION}b.c
```

대괄호는 b문자렬앞에서 문자 VERSION과 구별해주어 새로운 이름 VERSIONb가 생기는것을 막는다.

### 3.5.2. 지령확장

만일 표준출구를 환경변수로 할당하는 프로그람을 실행하려면 이름을 쓰지 않고 프로그람을 실행하도록 쉘에 지시하여야 한다. 그리고 현재 등록가입한 사용자의 목록을 환경변수로 할당하려면 다음과 같이 하여서는 안된다.

ULIST=who　**틀림**

이 지령의 결과는 단순히 who이다.

두개의 력점사이에 문자렬을 넣어야 쉘이 사용자가 요구하는 지령을 실행할수 있다.

ULIST='who'

아래와 같이 지령행의 인수도 포함할수 있다.

ULIST='who -q'

같은 기능을 수행하는 또다른 기호가 있는데 그것은 $기호와 괄호( )를 함께 쓰는것이다.

```
ULIST=$(who -q)
```

### 3.5.3. 대괄호확장

대괄호{ }를 써서 특수한 이름목록을 만들수 있다.

실례로 ccNNNj.doc형태의 서식을 가진 파일들로 이루어진 등록부가 있다고 하자.

여기서 NNN은 3개의 수자를 의미한다. 만일 수자가 210, 399, 410인 파일들의 크기와 생성날자를 얻으려고 한다면 다음과 같은 지령을 입력한다.

ls -l cc{210,399,410}j.doc

쉘은 다음과 같이 지령행을 확장할수 있다.

ls -l cc210j.doc cc399j.doc cc410j.doc

또한 다음의 지령은 등록부들을 만든다.

mkdir /home/kim/work{current,previous,next,safety}

### 3.5.4. tilde확장

물결표(~) 혹은 물결표와 함께 쓴 사선(/)은 현재 등록가입한 사용자의 home등록부의 경로이름과 동일하다. 체계의 어디서든지 다음의 지령으로 홈등록부에 있는 파일의 목록을 볼수 있다.

ls ~

홈등록부에 있는 부분등록부의 내용을 보려면 일반적으로 다음과 같이 하면 된다.

ls ~/result/workdir

물결표(~)기호는 다른 사용자의 홈등록부의 주소로써도 사용할수 있다.

물결표는 /etc/passwd에 있는 사용자의 이름에 의하여 사용자의 홈등록부의 이름으로 확장된다. 실례로 /usr/games를 자기의 홈등록부로 가지고 등록가입할 때 그 등록부들을 보기 위해서는 다음의 지령을 쓸수 있다.

ls ~games

## 제6절. 쉘함수

함수는 지령 혹은 지령묶음에 대한 이름이다. 함수들은 프로그람을 모듈화함으로써 효률적으로 작성할수 있게 한다. 그것들은 현재 쉘의 환경에서 실행된다. 사용자는 함수들을 다른 파일에 기억시키고 필요하면 그것들을 스크립트에서 호출하여 사용할수 있다.

함수에 관한 몇가지 중요한 규칙들은 다음과 같다.

① 쉘에서는 사용자가 입력한 내용이 별명, 함수, 내부지령 혹은 디스크에 있는 실행프로그람(또는 스크립트)중에서 어느것인가를 판단하는 과정을 거친다. 판단은 별명, 함수, 내부지령, 실행파일과 같은 순서로 진행된다.

② 함수는 그것을 리용하기전에 반드시 정의하여야 한다.

③ 함수는 현재 환경에서 실행된다. 함수는 자기를 호출한 스크립트와 변수를 공유할뿐아니라 인자 역시 위치파라메터로 지정하여 함수에 전달할수 있다. local함수를 써서 국부변수들을 함수내부에 만들수 있다.

④ 함수내부에서 exit지령을 실행하면 함수에서 탈퇴하여 스크립트안의 그 함수를 호출한 곳으로 돌아간다

⑤ 함수에서 사용한 return명령문은 함수안에서 마지막으로 실행된 지령의 탈퇴상태 또는 지정한 인수의 값을 돌려준다.

⑥ 내부지령 export -f를 써서 자식쉘에서도 함수를 그대로 사용할수 있다.

⑦ declare -f지령을 사용하여 함수의 이름과 그 정의내용을 현시할수 있다. 함수이름을 정확히 현시하자면 declare -F를 리용한다.

⑧ trap는 변수와 같이 스크립트와 스크립트에서 호출된 여러 함수들에서 공유할수 있다. trap가 함수에서 정의되면 그것은 스크립트에 의해서도 공유된다.

⑨ 함수가 다른 파일에 보관되여있으면 source나 dot지령으로 현재스크립트에 그 함수들을 적재할수 있다.
⑩ 함수가 재귀함수일수도 있다. 즉 함수는 자기자체를 호출할수 있다. 재귀호출회수에는 제한이 없다.

| 형식 |
| --- |
| function 함수이름 { 지령들; 지령들; } |
| **실례 3-19.** |
| function dir { echo "Directories: "; ls -l\|awk '/^d/ {print $NF}'; } |
| **설명** |
| function다음에는 함수이름 dir가 놓인다.(또한 빈 괄호가 함수이름뒤에 올수 있다.) 대괄호안의 지령들은 dir를 입력할 때 실행된다. 이 함수의 기능은 현재 작업등록부아래의 보조등록부를 현시하는것이다. 첫번째 대괄호의 앞뒤에는 공백을 주어야 한다. |

**- 함수설정해제**
unset지령을 사용하면 함수를 기억기에서 제거할수 있다.

| 형식 |
| --- |
| unset -f function_name |

**- 함수의 반출(export)**
함수들을 반출하여 자식쉘에서 그 함수들을 리용할수 있게 한다.

| 형식 |
| --- |
| export -f 함수이름 |

### 3.6.1. 함수의 인수와 되돌이값

함수는 현재쉘에서 실행되므로 변수들은 함수와 쉘에서 다 사용할수 있다. 함수에서의 사용자환경변화는 쉘에도 그대로 전달된다.

**- 인수**
위치파라메터를 써서 함수에 인수를 넘겨줄수 있다. 함수에서 사용하는 위치파라메터는 그 함수에 국한된다. 즉 함수의 인수들은 함수밖에서 리용되는 다른 위치파라메터들에 영향을 주지 않는다.(실례 3-20 참고)

**- 내부지령 local**
이 지령은 한 함수에만 국한되며 그 함수가 완료되면 없어지게 되는 국부변수들을 만드는데 사용한다.(실례 3-21 참고)

**- 내부지령 return**
이 지령은 함수에서 탈퇴하여 함수를 호출한 곳에서 프로그람에 조종을 돌려줄 때 쓴다. 함수의 되돌이값은 사용자가 return지령에 특정한 인수를 주지 않는 경우에는 스크립트에 있는 마지막지령의 탈퇴상태값과 같다. return지령에 값을 대입한다면 그 값은 0부터 255사이의 옹근수값만을 가지는 ?변수에 기억된다. return지령의 되돌이값은

0부터 255사이의 옹근수값만을 가지므로 함수의 출력을 되돌이값으로 리용할수 없는 경우에는 지령치환(command substitution)을 리용하여 출력을 포착(capture)한다. $(function_name)과 같이 화페기호가 앞에 붙은 괄호안에 함수전체를 넣어 UNIX지령의 출력을 얻을 때와 같이 출력을 특정한 변수에 저장한다.

**실례 3-20.**

(스크립트)

```
# !/bin/bash
# Scriptname: checker
# Purpose: Demonstrate function and arguments
1   function Usage { echo "error: $*" 2>&1; exit 1; }
2   if (( $# != 2 ))
    then
3       Usage "$0: requires two arguments"
    fi
4   if [[ !(-r $1 && -w $1) ]]
    then
5       Usage "$1: not readable and writeable"
    fi
6   echo The arguments are: $*
    < Program continues here >
```

(실행결과)

```
$ checker
error: checker: requirs two arguments
$ checker file1 file2
error: file1: not readable and writeable
$ checker filex file2
The arguments are filex file2
```

**설명**

1 Usage라는 함수를 정의한다. 이 함수는 표준오유화면에 오유통보문을 보내는 함수이다. 함수의 인수들은 함수를 호출할 때 여러개의 문자렬로 되여있다. 모든 위치파라메터들은 특정한 변수인 $*에 기억된다. 함수안의 파라메터들은 국부적이며 함수밖에서 쓰이는 위치파라메터들에 영향을 주지 않는다.

2 지령행에서 스크립트에 넘어가는 인수가 2개가 아니면 프로그람의 조종은 3행으로 넘어간다.

3 Usage함수를 호출할 때 문자렬 $0: requires two arguments를 인수로서 함수에 전달하며 변수 $*에 기억된다. echo지령은 이 문자렬통보문을 표준오유화면에로 내보내고 1행의 완료상태값을 되돌려주며 함수는 완료된다.

4,5 지령행에서 지정한 첫번째 인수가 실행 및 읽기가능한 파일의 이름이 아니라면

Usage함수는 $1: not readable and writeable을 인수로 가진다.
6 지령행에서 스크립트에 넘어가는 인수들은 변수 $*에 저장된다. 이 변수는 함수에 있는 $*에 아무런 영향을 주지 않는다.

**실례 3-21.(return지령을 리용한 스크립트)**

```
$ cat do_increment
# !/bin/bash
# Scriptname: do_increment
1  increment ( )  {
2      local sum;      # sum is known only in this function
3      let "sum=$1 + 1"
4      return $sum      # Return the value of sum to the script.
   }
5  echo -n "The sum is "
6  increment 5      # Call function increment; pass 5 as a
                    # parameter. 5 becomes $1 for the increment
                    # function.
7  echo $?          # The return value is stored, in $?
8.  echo $sum       # The variable "sum" is not known here

(실행결과)
$ do_increment
The sum is 6
```

**설명**

1 increment함수를 정의한다.
2 내부지령 local로 국부변수 sum을 정의한다. 이 변수는 함수밖에서는 사용되지 않으며 함수가 완료되면 제거된다.
3 함수가 호출되면 함수의 첫 인수 $1의 값은 하나 증가되며 결과값은 sum에 저장된다.
4 내부지령 return에 인수가 지정되면 함수를 호출한 다음행으로 조종이 넘어간다. 인수는 ?변수에 저장된다.
5 화면에 문자렬이 현시된다.
6 increment함수는 인수를 5로 하여 호출된다.
7 함수가 완료되면 그의 완료상태는 ?변수에 저장된다. return지령에서 인수를 지정하지 않으면 함수에서 실행된 제일 마지막지령의 완료상태가 되돌려진다. return지령의 인수는 반드시 0부터 255사이의 값이여야 한다.
8 변수 sum이 함수 increment에서 정의되였지만 국부변수이므로 함수밖에서는 사용할수 없으며 결과 아무것도 현시되지 않는다.

**실례 3-22.**(지령치환을 사용한 스크립트)

```
$ cat do_square
#!/bin/bash
# Scriptname: do_square
1   function square {
        local sq        # sq는 함수의 국부변수
        let "sq=$1 * $1"
        echo "$1는 두제곱하려는 수입니다."
2   echo "계산결과는 $sq입니다."
    }
3   echo "두제곱하려는 수를 입력하시오."
    read number
4   value_returned=$(square $number)    # 지령치환 Command substitution
5   echo    "$value_returned"
(실행결과)
$ do_square
3  두제곱하려는 수를 입력하시오.
    10
5  10은 두제곱하려는 수이다.
    계산결과는 100입니다.
```

**설명**

1 square함수를 정의한다. 이 함수는 인수 $1을 두제곱하는 함수이다.
2 수를 두제곱한 결과를 출력한다.
3 사용자입력을 요구한다.
4 square함수는 사용자가 입력한 수를 인수로 하여 호출된다. 함수가 $가 앞에 붙은 괄호안에 있기때문에 지령치환이 진행된다. 2개의 echo명령문들이 함수의 출력으로서 변수 value_returned에 대입한다.
5 지령치환과정에 계산된 값이 현시된다.

### 3.6.2. 함수와 source(또는 dot)지령

**- 함수의 저장**

함수를 반출할수도 있고 파일에 저장할수도 있다. 저장해둔 함수가 필요한 경우에 source 혹은 dot지령을 함수가 저장된 파일이름과 함께 입력하면 그 함수의 정의가 다시 효력을 가진다.

**실례 3-23.**

```
1   $ cat myfunctions
2   function go( ) {
        cd $HOME/bin/prog
        PS1 = [\h@\u\w]\$
```

```
        ls
    }
3   function greetings( ) { echo " $1 안녕하십니까! "; }
4   $ source myfunctions
5   $ greetings 명철동무
(실행결과)
명철동무 안녕하십니까!
```

**설명**

1 myfunctions파일은 2개의 함수(go와 greetings)를 정의하고있다.
2 첫번째로 정의된 함수는 go이다. 이 함수는 1차재촉문을 현재 작업등록부로 설정한다.
3 두번째로 정의된 함수는 greetings이다. 이 함수는 사용자의 이름을 인수로 넘겨받아 인사말을 현시한다.
4 source 또는 dot지령은 myfunctions의 내용을 쉘기억기에 적재한다.
5 greetings함수가 호출되여 실행된다.

**실례 3-24.**

```
1   $ cat dbfunctions
2   function addon ( ) {   # Function defined in file dbfunctions
3       while true
        do
            echo "Adding information "
            echo "Type the full name of employee "
        read name
        echo "Type address for employee "
        read address
        echo "Type start date for employee (4/10/08 ) :"
        read startdate
        echo $name:$address:$startdate
        echo -n "Is this correct? "
        read ans

        case "$ans"  in
            [Yy]*)
                echo "Adding info..."
                echo $name:$address $startdate--datafile
                sort -u datafile -o datafile
                echo -n "Do you want to go back to the main menu?
                read ans
                if [[ $ans ==[Yy] ]]
```

```
              then
4                 return        # Return to calling program
              else
5                 continue    # Go to the top of the loop
              fi
           ;;
           *)
              echo "Do you want to try again? "
              read answer
              case "$answer" in
                 [Yy]*) continue;;
                 *) exit;;
              esac
           ;;
        esac
     done
6   }     # End of function definition

(지령행)
7   $ more mainprog
    #!/bin/bash
    # Scriptname: mainprog
    # This is the main script that will call the function, addon
    datafile=$HOME/bourne/datafile
8   source dbfunctions    # The file is loaded into memory
    if [ ! -e $datafile ]
    then
       echo "$(basename $datafile) does not exist" >&2
       exit 1
    fi
9   echo "select one:"
    cat <<EOF
        [1] Add info
        [2] Delete  info
        [3] Update  info
        [4] Exit
    EOF
    case $choice in
       1) addon        # Calling the addon function
```

```
          ;;
        2) delete        # Calling the delete function
          ;;
        3) update
          ;;
        4)
          echo Bye
          exit 0
          ;;
        *) echo Bad choice
          exit 2
          ;;
    esac
    echo Returned from function call
    echo The name is $name
    # Variable set in the function are known in this shell.
    done
```

**설명**

dbfunctions파일에는 주프로그람에 쓰인 함수들이 포함되여있다.

1 dbfunctions파일이 현시된다.
2 addon함수가 정의된다. 이 함수는 datafile에 새로운 정보를 추가하는것이다.
3 while순환이 시작된다. break나 continue와 같은 명령문이 순환내부에 없으면 무한순환이 진행된다.
4 return지령은 함수를 호출한 프로그람에로 조종을 돌려보낸다.
5 조종은 while순환의 시작부분에로 돌아간다.
6 대괄호({})는 함수정의의 끝을 의미한다.
7 이 부분이 기본스크립트이다. addon함수가 스크립트에서 리용된다.
8 source지령은 dbfunctions파일을 쉘기억기에 적재한다. 함수 addon이 정의되고 리용가능하게 된다. 이것은 마치 스크립트의 이 부분에서 함수를 정의한것과 같다.
9 here문서를 리용하여 차림표를 구성한다. 사용자에게 차림표항목을 선택할것을 요구한다.
10 addon함수가 호출된다.

### 3.6.3. 신호처리

프로그람이 실행될 때 Ctrl+C 또는 Ctrl+¥를 누르면 그 신호가 도착하자마자 프로그람이 끝난다. 신호가 도착하자마자 즉시에 끝내지 않도록 할 필요가 있다. 다시말하여 신호를 무시하고 프로그람이 계속 실행되게 하던가 또는 스크립트에서 탈퇴하기전에 몇가지 작업을 수행할 필요가 있다. trap지령은 프로그람이 신호를 받을 때 동작하는 방식을 조종한다.

trap지령은 신호를 받으면 현재 실행중에 있는 쉘지령을 끝낼것을 요구한다. trap지령다음에 몇개의 지령을 인용부호안에 넣어 입력하면 특정한 신호에 대하여 지정한 지령들이 실행된다. 쉘은 trap지령 다음의 지령문자렬을 두번 즉 trap가 설정될 때와 신호가 도착할 때 읽는다. 이때 지령문자렬을 2중인용부호안에 넣어 입력하면 모든 변수와 지령치환은 trap가 설정되기전에 먼저 진행된다. 지령문자렬이 단일인용부호안에 있다면 신호가 검출되여 trap가 실행될 때까지 변수와 지령치환은 진행되지 않는다.

지령 kill -l 또는 trap -l을 리용하여 모든 신호들의 종류를 알아볼수 있다.(표 3-9)

**형식**

trap '지령;지령' 신호번호
trap '지령;지령' 신호이름

**실례 3-25.**

```
trap 'rm tmp*;exit 1' 0 1 2 15
trap 'rm tmp*;exit 1' EXIT HUP INT TERM
```

**설명**

신호 1(정지:hangup), 2(새치기:interrupt), 15(프로그람 끝:software termination) 가운데서 임의의 신호가 도착하면 모든 tmp파일을 삭제하고 탈퇴한다.

스크립트실행중에 새치기가 들어오면 trap지령은 사용자가 여러가지 방법으로 새치기신호를 조종하도록 한다. 즉 신호를 정상(표준)으로 처리하든가 무시하든가 또는 적합한 새치기조종기를 만들수 있다. HUP와 INT와 같은 신호이름들앞에 보통 SIGHUP, SIGINT 등과 같이 SIG가 붙는다. bash쉘에서는 앞붙이 SIG를 가지지 않는 신호에 대해 기호이름(symbolic name)을 리용하거나 신호들에 대해 수값을 리용할수 있다. 쉘이 탈퇴할 때 허위신호이름 EXIT 또는 수자 0을 사용하면 trap가 실행된다.

표 3-9. 신호번호와 종류(kill-1)

| | | | |
|---|---|---|---|
| 1) SIGHUP | 9) SIGKIL | 18) SIGCONT | 26) SIGVTALRM |
| 2) SIGINT | 10) SIGUSR1 | 19) SIGSTOP | 27) SIGPROF |
| 3) SIGQUIT | 11) SIGSEGV | 20) SIGSTP | 28) SIGWINCH |
| 4) SIGILL | 12) SIGUSR2 | 21) SIGTTIN | 29) SIGIO |
| 5) SIGTRAP | 13) SIGPIPE | 22) SIGTTOU | 30) SIGPWR |
| 6) SIGABRT | 14) SIGALRM | 23) SIGURG | |
| 7) SIGBUS | 15) SIGTERM | 24) SIGXCPU | |
| 8) SIGFPE | 17) SIGCHLD | 25) SIGXFSZ | |

**- 신호의 재설정**

신호를 기정방식으로 재설정하려면 trap지령뒤에 신호이름이나 번호를 덧붙이면 된

다. 함수에서 설정된 trap는 함수가 호출되면 그 함수를 호출한 쉘에서도 인식된다. 함수밖에서 설정된 trap역시 함수안에서 사용된다.

| 실례 3-26. |
|---|
| trap 2 또는 trap INT |
| 설명 |
| 2번신호인 SIGINT를 초기값으로 재설정한다. 이 신호는 Ctrl+C를 누를 때처럼 프로쎄스를 끝낸다. |

**- 신호의 무시**

trap지령다음에 한쌍의 빈 괄호가 놓이면 그 뒤에 렬거된 신호는 무시된다.

| 실례 3-27. |
|---|
| trap "" 1 2 또는 trap "" HUP INT |
| 설명 |
| 1번신호 SIGHUP과 2번신호 SIGINT는 쉘프로쎄스에 의해 무시된다. |

**- trap상태의 현시**

trap를 입력하면 trap에 지정된 모든 내용이 현시된다.

**실례 3-28.**

```
1   $ trap 'echo "Caught ya!; exit"' 2
2   $ trap
    trap - 'echo "Caught ya!; exit 1"' SIGINT
3   $ trap -
```

**설명**

1 trap지령은 2번신호(Ctrl+C)를 받을 때 스크립트를 완료하도록 설정된다.
2 인수가 없는 trap지령은 설정된 모든 trap들을 현시한다.
3 인수가 가로선(-)이면 쉘이 기동하였을 때 모든 신호들을 원래 가지고있던 초기값으로 재설정한다.

**실례 3-29.**

(스크립트)

```
# !/bin/bash
# Scriptname: trapping
# Script to illustrate the trap command and signals
```

```
# Can use the signal numbers or bash abbreviations seen
# below. Cannot use SIGNIT, SIGQUIT, etc.
1  trap 'echo "Ctrl+C will not terminate $0." ' INT
2  trap 'echo "Ctrl+\ will not terminate $0." ' QUIT
3  trap 'echo "Ctrl+Z will not terminate $0." ' TSTP
4  echo "Enter any string after the prompt.
   When you are ready to exit, type \"stop\"."
5  while true
   do
6     echo -n "Go ahead... > "
7     read
8     if [[ $REPLY== [Ss]top ]]
      then
9        break
      fi
10 done
```

(실행결과)

```
   $ trapping
4   Enter any string after the prompt.
    When you are ready to exit, type "stop".
6   Go ahead...> (ctrl+c를 누른다)
1   Ctrl+C will not terminate trapping.
6   Go ahead...> (ctrl+z를 누른다)
3   Ctrl+Z will not terminate trapping.
6   Go ahead... > (ctrl+\를 누른다)
2   Ctrl+\ will not terminate trapping.
6   Go ahead... > stop
```

설명

1 첫번째 trap는 INT신호(Ctrl+C)를 검출한다. 프로그람의 수행과정에 Ctrl+C를 누르면 인용부호안에 있는 지령이 실행된다. 프로그람은 중지되지 않고 “Ctrl+C will not terminate trapping”이라는 통보문을 현시하고 사용자입력을 기다린다.

2 두번째 trap는 사용자가 QUIT신호인 Ctrl+\을 누를 때 실행된다. 문자렬 “Ctrl+\ will not terminate traspping”이 현시되고 프로그람이 계속 실행된다. 원래 이 신호는 프로쎄스를 끝내고 core파일을 작성하는 역할을 수행한다.

3 세번째 trap는 사용자가 TSTP신호인 Ctrl+Z를 누를 때 실행된다. 문자렬 “Ctrl+Z will not terminate trapping”이 현시되고 프로그람이 계속 실행된다. 보통 이 신호는 일감조종이 가능한 경우 배경에서 동작하고있는 프로그람의 실행을

중단시킨다.
4 사용자입력을 요구한다.
5 while순환이 시작된다.
6 문자렬 "Go ahead...>"를 현시한 후 사용자입력을 기다린다.
7 read지령은 내부변수 REPLY에 사용자입력값을 대입한다.
8 REPLY값이 Stop 혹은 stop라면 break지령에 의해 순환에서 탈퇴하고 프로그람을 끝낸다. Stop나 stop를 입력해야만 이 프로그람에서 탈퇴할수 있다. 이 방법은 kill지령으로 프로그람에서 탈퇴하는것과 같다.
9 break지령은 순환에서 탈퇴하게 한다.
10 done예약어는 순환의 끝을 표시한다.

**- 함수안에서 trap**

사용자가 trap를 리용하여 함수에서 신호를 처리하면 함수가 호출될 때 그것은 전체 스크립트에 영향을 준다. 즉 trap는 스크립트에 적용된다. 다음 실례에서 새치기건 ^C를 무시하도록 설정하였다. 이 스크립트에서 순환을 중지하려면 kill지령을 리용하여야 한다. 이 실례는 함수안에서 trap를 실행할 때 생길수 있는 문제점을 보여준다.

**실례 3-30**

(스크립트)

```
    #!/bin/bash
1   function trapper ( ) {
        echo "In trapper"
2       trap 'echo "Caught in a trap!"' INT
        # Once set, this trap affects the entire script. Anytime
        # ^C is entered, the script will ignore it.
    }
3   while :
    do
        echo "In the main script"
4       trapper
5       echo "Still in main"
        sleep 5
    done
```

(실행결과)

```
$ trapper
In the main script
In trapper
Still in main
^CCaught in a trap!
```

In the main script

**설명**

1 trapper함수가 정의된다. 이 함수의 모든 변수와 trap들은 스크립트에서도 사용할 수 있다.
2 trap지령은 새치기건(^C)인 2번신호 INT를 무시한다. ^C를 누르면 "Caught in a trap"와 같은 통보문이 현시되며 스크립트는 계속 실행된다. 스크립트는 Ctrl+z로 중지시킬수 있다.
3 기본스크립트는 무한순환에 빠진다.
4 trapper함수를 호출한다.
5 함수가 완료되면 여기서부터 다시 시작된다.

## 제7절. 오유수정

가장 간단한 오유수정방법은 echo지령으로 결과값을 현시해보는 방법이다. 이 방법은 C언어에서 printf, Perl언어에서는 print지령을 사용하는것과 비슷하다. echo지령으로 변수값을 현시해보므로써 어느 부분에서 실수했는가를 알수 있다. 대부분의 쉘프로그람작성자는 이러한 오유를 찾는데 전체 프로그람작성시간의 80%를 소비한다고 한다. 쉘스크립트는 echo지령으로 오유를 수정하는 과정에 다시 콤파일할 필요가 없으므로 시간을 절약할수 있다는 우점을 가지고있다.

쉘스크립트에서 오유수정방법은 다음과 같다.

bash -x strangescript

strangescript는 오유를 수정하려고 하는 쉘스크립트의 이름이다. 우와 같이 스크립트를 실행하면 쉘이 실행되는 동안에 사용된 모든 변수의 값을 출력시켜준다. 그러므로 어느 곳에서 실수를 했는지 쉽게 찾을수 있다.

bash지령에 -n추가선택을 리용하면 지령들을 실행시켜보지 않고도 문법오유를 찾아낼수 있다.

bash -n strangescript

즉 스크립트에 문법오유가 있으면 쉘이 오유를 알려준다. 오유가 없으면 아무것도 현시하지 않는다.

스크립트의 오유를 수정하는데 널리 리용되는 지령은 -x추가선택을 가진 set지령과 -x추가선택과 스크립트이름으로 호출되는 bash지령이다. 표 3-10에 오유수정추가선택들을 보여주었다. 이 추가선택들은 스크립트의 실행을 추적할수 있게 한다. 스크립트의 매 지령은 치환이 진행된 후 현시되며 그 다음에 실행된다. verbose추가선택이 선택되거나 -v추가선택(bash –v 스크립트)으로 쉘을 호출하면 스크립트의 매 행이 입력된 그대로 현시되고 그 다음에 실행된다. 스크립트안의 행이 현시될 때 그 앞에 더하기(+)기호가 붙는다.

표 3-10. 오유수정 추가선택

| 지 령 | 추가선택 | 사 명 |
|---|---|---|
| bash -x scriptname | echo추가선택 | 스크립트의 매개 행을 변수치환하여 현시한 후 실행한다. |
| bash -v scriptname | verbase추가선택 | 스크립트의 매개 행을 그대로 현시한 후 실행한다. |
| bash -n scriptname | noexec추가선택 | 지령을 해석하지만 실행하지는 않는다. |
| set -x | echo켜기 | 스크립트에서 실행을 추적한다. |
| set +x | echo끄기 | 추적을 해제한다. |

다음의 실례는 todebug스크립트에 대한 오유수정과정을 보여준다.

**실례 3-31.**

(스크립트)

```
$ cat todebug
#!/bin/bash
# Scriptname: todebug
1   name="Joe Shmoe"
    if [[ $name == "Joe Blow" ]]
    then
        printf "Hello $name\n"
    fi
    declare -i num=1
    while (( num<5 ))
    do
        let num+=1
    done
    printf  "The total is %d\n", $num
```

(실행결과)

```
2  bash -x  todebug
   + name=Joe Shmoe
   + [[ Joe Shmoe == \J\o\e\\B\l\o\w ]]
   + declare -i num=1
   + (( num<5 ))
   + let num+=1
   + (( num<5 ))
   + let num+=1
   + (( num<5 ))
   + let num+=1
```

```
+ (( num<5 ))
+ let num+=1
+ (( num<5 ))
+ printf "The total is %d\n," 5
The total is 5
```

**설명**

1 스크립트의 이름은 todebug이다. -x추가선택을 설정하여 스크립트가 실행하는것을 볼수 있다. 매 순환에서 프로쎄스가 현시되며 변수들의 값이 설정되거나 변경될 때 현시된다.
2 bash지령에 -x추가선택을 주어 시작한다. echo기능이 설정된다. 스크립트의 매 행의 앞에 더하기(+)기호가 붙여져 현시된다.

## 제8절. 내부 bash지령

bash쉘에는 상당히 많은 지령들이 있다.
쉘은 원천코드를 내장하고있는 지령들을 가지고있다. 지령들이 내장되여있기때문에 쉘은 디스크에서 호출할 때 보다 고속으로 실행된다. help지령은 임의의 내부지령에 대한 직결도움말을 보여준다. 내부지령목록을 표 3-11에 주었다.

표 3-11. 내부지령

| 지 령 | 기 능 |
|---|---|
| : | 아무것도 수행하지 않고 완료상태값을 0으로 되돌려준다. |
| .file | 점(.)지령은 파일을 읽어서 실행한다. |
| break[n] | 순환에서 탈퇴한다. |
| . | 현재 쉘안에서 프로그람을 실행한다. source지령과 같다. |
| alias | 별명목록을 현시하거나 새로 작성한다. |
| bg | 배경에 일감을 보낸다. |
| bind | readline함수나 마크로에 대한 현재 건과 함수묶음들, 묶음건을 현시한다. |
| break | 제일 안쪽의 순환을 중지한다. |
| builtin[sh- builtin [arg]] | 인수들을 전달하고 완료상태값을 0으로 되돌려주면서 쉘내부지령을 실행한다. 보통 함수와 내부지령의 이름이 같을 때 적용한다. |
| cd[arg] | 인수 arg가 없으면 home등록부에로, 그렇지 않으면 arg의 값으로 변경된다. |

표계속

| 지　령 | 기　능 |
| --- | --- |
| command, command[arg] | 함수와 동일한 이름으로 되여있어도 지령을 실행한다.(함수를 찾아보는 과정이 생략된다.) |
| continue[n] | 순환이 계속된다. |
| declare[var] | 모든 변수를 현시하거나 또는 어떤 속성을 지정하여 변수를 선언한다. |
| dirs | pushd로 얻어진 현재등록부의 목록을 현시한다. |
| disown | 일감표로부터 능동일감을 제거한다. |
| echo[args] | 변수값을 현시한다. |
| enable | 쉘내부지령을 유효 또는 무효로 한다. |
| eval[args] | 실행전에 지령행의 내용을 두번 검사하게 한다. 한번은 쉘이 인수로 해석하고 다음은 지령으로 실행한다. |
| exec command | 현재 쉘위치에서 지령을 실행한다. |
| exit[n] | 완료상태값을 n으로 되돌려주고 쉘을 완료한다. |
| export[var] | 자식쉘에게 var를 넘겨준다. |
| fc | 리력목록을 편집하기 위한 지령이다. |
| fg | 전경에 배경일감을 넣는다. |
| Getopts | 지령행에서 지정한 유효한 추가선택을 추출하기 위해 스크립트에서 사용한다. |
| hash | 지령들을 고속실행하기 위해 내부하쉬표를 조종한다. |
| Help[command] | 내부지령에 대한 도움말을 제공한다 |
| history | 행번호와 함께 리력목록을 현시한다. |
| Jobs | 배경의 일감들을 현시한다. |
| kill[-signal process] | 지정한 PID번호나 혹은 일감번호에 신호를 보낸다. |
| let | 산수연산표현식의 값을 계산하고 변수에 산수연산계산값을 대입한다. |
| local | 변수의 적용범위를 함수내부로 제한하는데 사용한다. |
| logout | 등록가입쉘을 완료한다. |
| popd | 등록부탄창에서 항목을 제거한다. |
| pushd | 등록부탄창에 항목을 첨부한다. |
| pwd | 현재작업등록부를 출력한다. |
| read[var] | 표준입력으로부터 행을 읽어 인수 var에 대입한다. |

표계속

| 지　　령 | 기　　능 |
|---|---|
| readonly[var] | 변수 var를 읽기전용으로 만든다. 변수 var를 다시 설정할수 없다. |
| return[n] | 완료상태값을 n으로 되돌려주고 함수를 완료한다. |
| set | 추가선택들과 위치인수를 설정한다. |
| shift[n] | 왼쪽으로 n번 위치인수를 옮긴다. |
| stop pid | 지정한 PID의 실행을 중지한다. |
| suspend | 현재 쉘의 실행을 일시 중지한다. |
| test | 파일형태들을 검사하고 조건식을 평가한다. |
| times | 현재 쉘로부터 실행되는 프로쎄스들의 실행시간에 대한 정보를 출력한다. |
| trap[arg][n] | 쉘이 신호 n(0, 1, 2 혹은 15)을 받을 때 arg를 실행한다. |
| Type[command] | 지령의 형태를 출력한다. |
| typeset | declare와 같다. 변수들을 대입하고 속성들을 준다. |
| ulimit | 프로쎄스가 사용할수 있는 자원의 최대한계를 설정한다. |
| umask[octal digits] | 파일생성시에 적용할 허가권한(소유자, 그룹, 기타)을 설정한다. |
| unalias | 별명들을 해제한다. |
| unset[name] | 변수나 함수의 설정을 해제한다. |
| wait[pid #n] | PID번호가 n인 배경프로쎄스가 끝날 때까지 스크립트를 일시 중지한다. |

## 제9절. 편리한 지령행편의프로그람

이 절에서 취급하게 되는 몇가지 지령행편의프로그람들을 리용하여 사용자는 가상말단을 설정하고 변경시킬수 있으며 파일에 대한 정보와 형태, 내용검사 등 여러가지 작업을 효과적으로 수행할수 있다.

### 3.9.1 가상말단변경(virtual terminal changing)

**가상말단**

다양한 특성을 가진 여러가지 말단장치들의 표준기능을 일반화한 론리적모형으로서 OSI환경에서 동작하도록 기능을 제공하는 접속규약(protocol)이다. 여기서 OSI(Open System Interconnection: 열린체계상호접속)는 분산되여있는 여러 기종의 콤퓨터들사이에 자료통신을 위해서 국제표준화기구 ISO와 국제전신전화자문위원회 CCITT가 제안한 망구성방식이다.

하나의 말단에서 동시에 여러번의 등록가입을 할수 있다. 실지 기능건 Alt+Ctrl+F1~Alt+Ctrl+F12까지 사용할수 있다. 가상말단의 기정번호는 1이다. 매 가상말단들은 서로 완전히 독립적이며 가상말단을 통하여 서로 다른 사용자들이 각이한 접근준위로써 등록가입할수 있다. 기정으로 가상말단은 6개로 설정되여있다.

한 가상말단에서 다른 가상말단으로 절환하여 현시하는 방법에는 2가지가 있다. 다음의 지령은 가상말단 3으로 절환한다.

chvt 3

Ctrl+Alt+F3으로도 가상말단 3으로 절환할수 있다.

사용과정에 가상말단의 개수가 12보다 작은 경우도 있다. 이것은 가입한 사용자수가 적거나 사용자들이 가상말단을 얼마 사용하지 않기때문이다. 가상말단의 수를 좀 더 늘이려면 /etc/inittab파일을 편집하고 그것을 목록에 첨부하여야 한다.

등록가입대기는 mingetty라는 이름을 가진 프로그람에 의하여 발생한다. 매 가상말단용 inttab파일이 번호별로 존재하므로 필요한만큼 이 파일을 복사하여 추가하려는 말단번호로 변경시켜야 한다.

### 3.9.2 파일정보탐색

- ls지령

디스크에 저장되여있는 파일에 대한 정보를 알아보는 몇가지 지령가운데서 가장 많이 사용하는것이 바로 파일이름을 현시하는 ls지령이다. ls지령에 -l추가선택을 사용하면 한 행에 한개 파일의 이름과 그 파일에 대한 일부 정보들을 현시한다.

-rwxr-xr-x 1 kim devel 20480 Apr 9 17:22 sedit.tar

우의 실례는 파일의 이름은 sedit.tar이고 4월 9일 오후(17시22분)에 작성 혹은 마지막으로 변경되였으며 파일의 총용량은 20480byte, devel그룹성원인 파일의 소유자는 kim이며 파일은 오직 한개의 하드와만 련결된다는것을 보여준다.

허가권부분의 문자들은 파일형태를 가리킨다.

실례로 -기호는 표준파일, d는 등록부, c는 문자장치, b는 블로크장치라는것을 가리킨다. 그러나 특수한 장치는 기본적으로 /dev등록부에만 있다.

허가권에는 3개의 모임이 있는데 개개는 3개의 문자렬 rwx로 이루어졌다. 문자가 없으면 -기호가 들어가는데 이것은 특별한 권한을 가지지 않는다는것을 의미한다. 만일 r가 설정되여있으면 읽기권한이 설정되며 w가 설정되면 쓰기권한, x가 설정되면 실행권한이 설정되여있다는것을 의미한다. 왼쪽끝의 3문자는 파일소유자에게 적용되고 가운데 3글자는 소유자그룹에 적용되며 오른쪽 3문자는 모든 사람에게 설정된다. 물론 특권사용자는 그 어떤 제한도 무시할수 있는 능력이 있다.

- type지령

쉘내부지령인 type는 입력한 지령이 어떤 동작을 하는가를 알려준다. 다음과 같은 type추가선택이 있다.

type -type cp

이것은 PATH환경변수에 있는 등록부에서 지령을 탐색하며 만일 찾으면 단순히 cp가 무엇인가를 현시해준다. 이것은 또한 파일이름이 별명, 실마리어, 함수, 내부지령중에서 어떤것인가를 가리킨다. 때로는 입력한 지령이 정말로 실행파일이 맞는가, PATH환경변수에 있는가 그렇지 않으면 어디에 있는가를 알아야 할 문제가 제기될수 있다. 이때는 type지령을 리용하여 간단히 해결할수 있다.

type -path cp
우와 같은 경우에는 전체 파일경로를 현시한다.

- file지령
이 지령은 파일의 형태를 결정한다.
file blogg
이 지령을 실행하면 blogg파일이 어떤 파일인가를 본문으로써 서술한다.
만일 파일이 프로그람이면 다음과 같이 서술된다.
ELF 32-bit LSB executable, Intel 80386, dynamically linked
또한 그것이 등록부라든지 ASCII본문이라든지 혹은 그 어떤 다른것이라고 알려줄수도 있다. 만일 file지령이 파일형태를 결정할수 없으면 2진자료라고 가정하고 다음과 같이 현시한다.
data
file지령은 수백개의 파일형태를 인식할수 있으며 새로운것도 쉽게 추가할수 있다.
/usr/share/file/magic에는 잘 알려진 파일형태목록과 파일들을 인식하기 위한 지령들이 있다. 대부분 파일들은 특수한 번호(magic number)로 볼수 있게 되여있는데 같은 형태의 파일은 같은 장소에 있다.

- state지령
이 지령은 지정한 파일에 대한 모든 정보를 즉시에 알려준다.
state /bin/grep
우와 같이 입력한 지령의 실행결과는 다음과 같다.

```
File: "/bin/grep"
Size: 69444            Filetype: Regular File
Mode: (0755/-rwxr-xr-x)        Uid: ( 0/ root)   Gid: ( 0/ root)
Device: 3, 0   Inode:16132      Links:1
Access: Fri Apr 16   12:23:45    1999(0000.00:10:28)
Modify: Wed  sep 9   23:13:46    2000(00218.13:20:27)
change: Thu Feb 4   05:10:27    2001(00071.06:23:46)
```

첫 두개행은 파일이름과 크기, 형태를 보여주며 다음행은 허가권과 사용자 ID와 사용자이름, 그룹 ID와 그룹이름을 보여준다. Device는 파일이 보관된 하드웨어장치이다. Inode는 Linux파일체계에서 마디번호로써 파일의 디스크주소와 이름을 가리킨다.

### 3.9.3 ASCII파일보기

cat지령을 사용하면 화면상에서 ASCII파일(본문파일)의 내용을 볼수 있다.
cat textfile
이 지령은 이름이 textfile인 본문파일의 내용을 현시하는데 그 내용이 한 화면을 넘어나면 마지막 24개의 행만을 볼수 있다.
more지령을 주면 한번에 본문내용을 한 화면(24행)씩 볼수 있다.
more textfile
Enter건을 누르면 한행씩, 공백건을 누르면 한 화면씩 다음의 내용을 현시하며 Q건을 누르면 탈퇴한다. 화면의 제일 밑에 지금까지 본 내용이 차지하는 프로수가 표시된다.

less지령을 쓰면 page up건과 page down건, 방향건을 눌러서 앞뒤로 자유롭게 이동하면서 본문내용을 볼수 있다.

less textfile

less지령에서도 more지령처럼 전체 화면에 본문을 현시하며 Enter건과 공백건, Q건의 기능도 같다. 문자 d(down: 아래방향)와 문자 u(up: 웃방향)를 방향건으로 리용할수도 있다. man지령을 사용할 때 less지령을 쓰면 문서의 본문내용을 현시할수 있다.

### 3.9.4 파일내용검사

grep편의프로그람을 리용하면 본문파일의 내용에서 필요한 문자렬을 찾을수 있다.

grep지령은 찾으려는 문자렬을 포함한 전체 행과 파일이름목록을 보여준다.

현재 등록부에서 .c로 끝나는 파일들을 찾고 거기서 문자렬 time이 들어간 행들을 현시하는 지령은 다음과 같다.

grep time *.c

정규표현식의 특수문자를 비롯하여 일반문자상수도 탐색문자렬로 될수 있다.

strings지령으로 2진파일 또는 알수 없는 형태의 파일에서도 필요한 모든 문자렬을 찾을수 있다.

strings /bin/grep

이 지령은 모든 본문통보문과 객체파일들의 이름을 현시한다. 한 행에 4개 또는 그 이상의 ASCII문자를 현시한다.

파일의 바이트값을 볼수 있는 2개의 편의프로그람이 있는데 하나는 od이고 다른 하나는 hexdump이다. od지령은 매 바이트의 값을 8진수로 보여주며 hexdump는 16진수로 보여준다.

실례로 이름이 fred인 본문파일에 "Use info to get more information"과 같은 본문이 포함되여있다고 하자.

od지령을 사용하여 이 파일을 읽으면 결과는 다음과 같다.

```
0000000 071525  020145  067151 067546 072040 020157 062547 020164
0000020 067555  062562  064440 063156 071157 060555 064564 067157
0000040 005056
```

hexdump지령으로 읽으면 다음과 같다.

```
0000000   7355   2065   6e69   6f66   7420   206f   6567   2074
0000010   6f6d   6572   6920   666e   726f   616d   6974   6e6f
0000020   0a2e
```

# 제4장. 쉘프로그람작성방법

쉘은 건반이나 파일로부터 지령을 읽어서 필요한 조작을 수행하는 프로그람이다. 지령행에서 실행할수 있는 수백여개의 지령들이 있다. 자주 쓰이는 지령들은 쉘프로그람내부에 있다. 이 여러개의 지령들을 조합하여 어떤 작업을 수행하도록 프로그람을 만들수 있다. 이렇게 만든 프로그람을 쉘프로그람이라고 하는데 쉘프로그람은 하나의 파일에 보관된 순차적인 여러가지 지령들이다. 이 파일은 실행가능하며 쉘수속 혹은 쉘스크립트 간단히 스크립트라고 부른다. 쉘프로그람은 순환과 조건부실행, 국부변수설정 등에 쓸수 있는 특수한 내부지령들로 이루어졌다.

Linux에서 쉘스크립트들은 콤파일되고 련결된 프로그람처럼 실행할수 있는 상태에 있다. 즉 해석형언어이다. 쉘스크립트들에는 실행권한이 주어져야 하며 실행하기 위해서는 이름을 입력해야 한다. 쉘프로그람은 자기가 입력한 프로그람을 읽어들이고 PATH 경로에 따라서 파일을 탐색하며 파일의 내용을 결정하고 만일 파일이 스크립트이면 지령을 해석하는 동작이 필요하다.

## 제1절. 쉘스크립트구조

쉘프로그람에는 여러가지가 있고 그것들의 문법이 각이하기때문에 어떤 쉘프로그람이 스크립트로써 실행되는가를 아는것이 필요하다. 실례로 C shell로 프로그람을 작성하고 bourn shell에서 실행한다면 스크립트는 실행되지 않는다.

이것을 해결하기 위하여 여러가지 쉘프로그람들은 특수한 구조로써 첫행에 이 스크립트가 어떤 쉘을 쓰는가 하는것을 알려준다.

례를 들면 다음과 같다.

```
#!/bin/sh
```

Linux의 경우 /bin/sh는 /bin/bash의 다른 이름으로써 이때 항상 bash프로그람으로써 실행되게 된다.

지어 사용자가 C shell로 실행하려고 해도 bash쉘은 스크립트를 실행하는데 호출되게 된다. 반대로 스크립트들이 항상 C shell로부터 실행되도록 하자면 첫 행이 다음과 같이 되여야 한다.

```
#!/bin/chsh
```

첫 행이 아닌 임의의 다른 행이 #로 시작되면 설명문으로 되여 뛰여넘기된다.

다음의 스크립트는 현재 등록가입한 모든 사용자와 주콤퓨터이름을 보여준다.

```
#!/bin/sh
hostname
who
exit 0
```

마지막행은 스크립트를 실행한 후에 되돌림값을 서술한다. 이것은 반드시 필요한것은 아니다.

스크립트를 실행하기전에 반드시 실행권한이 주어져야 한다. chmod지령은 파일에 실행권한을 부여하는데 사용된다. 실례로 파일이 가지고있는 스크립트이름이 wahoo이고 모든 사람이 실행할수 있게 권한을 주려면 다음과 같은 지령을 입력한다.

chmod 777 wahoo
777은 모든 사람들에게 읽기, 쓰기, 실행권한을 부여한다.
파일에 특정한 사용자만이 사용할수 있도록 권한을 부여할수도 있다.
실례로 자기의 그룹에서 다른 사용자는 수정하거나 삭제할수 없도록 권한을 부여하자면 다음과 같이 하면 된다.
chmod g+w wahoo
또한 그룹성원들에 대한 실행과 쓰기권한을 삭제할수도 있다.
chmod g-xw wahoo

## 제2절. 인수

쉘스크립트를 실행할 때 스크립트 다음에 있는 공백문자로 구분된 매 단어들을 인수라고 한다.

### 4.2.1. 위치파라메터

지령행인수는 위치파라메터로 스크립트안에서 참고될수 있다. 실례로 첫번째 인수는 $1, 두번째 인수는 $2, 세번째 인수는 $3 등으로 처리된다. $9다음의 인수들은 대괄호를 리용하여 수자부분을 묶어준다. 실례로 위치파라메터 10은 ${10}으로 참조한다. $# 변수는 파라메터들의 수를 검사하기 위하여 사용하며 $*는 그 모든것을 현시하기 위해 사용한다. 위치파라메터는 set지령에 의해 설정 또는 재설정할수 있다. set지령을 사용하면 이전의 설정값은 지워진다. 표 4-1에 위치파라메터를 보여주었다.

표 4-1. 위치파라메터

| 위치파라메터 | 의 미 |
|---|---|
| $0 | 현재쉘스크립트의 이름이다. |
| $# | 위치파라메터의 총개수를 표시한다. |
| $* | 모든 위치파라메터들을 표시한다. |
| $@ | 2중인용부호안에 있을 때를 제외하고는 $*와 같다. |
| "$*" | 단일인수로 확장한다.("$1 $2 $3"과 같다.) |
| "$@" | 인수를 분리하여 확장한다.("$1" "$2" "$3"과 같다.) |
| $1 $[10] | 개별적인 위치파라메터들을 참조한다. |

실례 4-1에서 위치파라메터에 대하여 보여주었다.

**실례 4-1.**

(스크립트)
```
# !/bin/bash
# Scriptname:greetings2
```

```
    echo "This script is called $0."
1   echo "$0  $1 and $2"
    echo "The number of positional parameters is $#"
      - - - - - - - - - - - - - - - - - - - - - - - - - - -
(지령행)
$ chmod +x greetings2
2  $ greetings2
   This script is called greetings2.
   greetings and
   The number of positional paramters is
3  $ greetings2 kim
   This script is called greetings2.
   greetings kim and
   The number of positional parameters is 1
4  $ greeting2 kim li
   This script is called greetings2
   greetings kim and li
   The number of positional parameters is 2
```

**설명**

1 스크립트 greeting2에서 위치파라메터 $0은 스크립트이름을 의미하며 $1은 첫번째 지령행인수 그리고 $2는 두번째 지령행인수를 의미한다.

2 greetings2스크립트는 아무런 인수들을 넘겨줌이 없이 실행한다. 출력은 greetings라는 스크립트와 $1과 $2에 아무것도 대입되지 않았다는것을 알수 있다. 그러므로 그 값들은 빈값이며 아무것도 현시되지 않는다. 스크립트이름은 $0에 보관된다.

3 이때 한개의 인수 kim을 넘겨준다. kim은 첫번째 위치파라메터에 대입된다.

4 2개의 인수 kim과 li를 입력시킨다. kim을 $1에 대입하며 li를 $2에 대입한다.

### 4.2.2. set지령과 위치파라메터

set지령에 인수를 주면 위치파라메터들이 다시 대입된다. 다시 대입하면 낡은 파라메터목록을 잃어버리게 된다. 모든 위치파라메터들을 해제하기 위해서는 set를 사용해야 한다. $0은 항상 스크립트의 이름이다. 실례 4-2에서 set지령과 위치파라메터의 쓰임에 대하여 보여주었다.

**실례 4-2.**

```
(스크립트)
#!/bin/bash
# Scriptname : args
# Script to test command line arguments
```

```
1  echo The name of this script is $0.
2  echo The arguments are $*.
3  echo The first argument is $1.
4  echo The second argument is $2.
5  echo The number of arguments is $#.
   oldargs=$*
6  set kim li pak     # Reset the positional parameters
7  echo All the positional parameters are $*.
8  echo The number of postional parameters is $#.
9  echo "Good-bye for now, $1."
10 set $(date)              # Reset the positional parameters
   echo The date is $2 $3, $6.
11 echo "The value of\$oldargs is $oldargs."
   set $oldargs
12 echo $1 $2 $3
```

(실행결과)

```
$ args a b c d
1  The name of this script is args.
2  The arguments are a b c d.
3  The first argument is a.
4  The second argument is b.
5  The number of arguments Is 4.
7  All the positional parameters are kim li pak.
8  The number of positional parameters is 3.
9  Good-bye for now, kim.
10 The date is Mar 25, 2006.
11 The value of $oldargs is a b c d.
12 a b c
```

**설명**

1 스크립트의 이름을 $0변수에 기억한다.

2 $*는 모든 위치파라메터들을 현시한다.

3 $1은 첫번째 위치파라메터를 현시한다.(지령행인수)

4 $2는 두번째 위치파라메터를 현시한다.(지령행인수)

5 $#는 위치파라메터들의 총수이다.(지령행인수들)

모든 위치파라메터들이 oldargs변수에 기억시킨다.

6 set지령은 낡은 목록들은 지우면서 위치파라메터들을 재설정하게 한다. 이때 $1은 kim, $2는 li 그리고 $3은 pak로 된다.

7 $*는 모든 파라메터들 즉 kim, li 그리고 pak을 현시한다.
8 $#는 파라메터의 수 즉 3을 나타낸다.
9 $1은 kim이다.
10 지령치환을 한 다음 date를 실행하고 위치파라메터들을 date지령의 출력으로 재설정한다. $2, $3 그리고 $6의 새로운 값들을 현시한다.
11 oldargs에 기억된 값들을 현시한다.
set지령은 oldargs안에 기억된 값들로 위치파라메터들을 만든다.
12 첫 3개의 위치파라메터들을 현시한다.

실례 4-3에서 위치파라메터와 특수변경자 :?에 대한 쓰임을 보여주었다.

**실례 4-3.**

(스크립트)

```
#!/bin/bash
# scrlptname:checker
# script to demonstrate the use of special variable
# modifiers and arguments
1  name=${1:?"requires an argument" }
   echo Hello $name
```

(지령행)

```
2  $ checker
   checker: 1: requires an argument
3  $ checker sue
   Hello sue
```

**설명**

1 특수한 변수변경자 :?는 $1이 값을 가지는가 안가지는가를 검사한다. 가지지 않으면 스크립트에서 탈퇴하고 통보문을 현시한다.
2 프로그람은 인수가 없이 실행되며 $1에는 값이 대입되지 않는다. 따라서 오유가 현시된다.
3 checker프로그람은 지령행인수 sue를 준다. 스크립트에서 $1에는 sue를 대입한다. 프로그람은 계속 실행된다.

**- $*와 $@의 차이**

$*와 $@는 오직 2중인용부호안에 있을 때만 차이난다. $*가 2중인용부호로 처리될 때 파라메터목록은 단일문자렬로 취급된다. $@가 2중인용부호안에 있을 때 매 파라메터들은 서로 분리된 문자렬로 처리된다. 다음의 실례 4-4를 통하여 $*와 $@의 쓰임과 차이에 대하여 보여주었다.

실례 4-4.

```
1   $ set 'apple pie' pears peaches
2   $ for i in $*
    > do
    > echo $i
    > done
    apple
    pie
    pears
    peaches
3   $ set 'apple pie' pears peaches
4   $ for i in "$*"
    > do
    > echo $i
    > done
    apple pie pears peaches
5   $ set 'apple pie' pears peaches
6   $ for i in $@
    > do
    > echo $i
    > done
    apple
    pie
    pears
    peaches
7   $ set 'apple pie' pears peaches
8   $ for i in "$@"
    > do
    > echo $i
    > done
    apple pie
    pears
    peaches
```

설명

1 위치파라메터들을 설정한다.

2 $*가 확장될 때 apple pie를 넣은 인용부호는 해제된다. 즉 apple과 pie는 2개의 분리된 단어들로 된다. for순환은 변수 i에 매 단어들을 대입하고 i의 값을 현시한다. 순환하면서 i의 값을 써넣어 매 단어들을 대입한다. 순환에 의해 매 단계마다 왼쪽에서 오

른쪽으로 자리밀기하며 다음 단어가 변수 i에 대입된다.
3 위치파라메터들을 설정한다.
4 2중인용부호안에 $*를 넣으면 전체 파라메터를 하나의 문자렬로 처리한다. 즉 전체 문자렬이 하나의 단어처럼 저장된다. 순환은 한번만 수행한다.
5 위치파라메터를 설정한다.
6 인용부호안에 넣지 않은 $@와 $*는 같은것으로 표현된다.
7 위치파라메터를 설정한다.
8 2중인용부호안에 $@를 넣으면 매 위치파라메터들은 한개 인용부호안에 넣은 문자렬로 처리된다. 목록은 "apple pie", "pears" 그리고 "peaches"로 구성된다.
위치파라메터들이 현시된다.

## 제3절. 조건부실행

### 4.3.1. 탈퇴상태

조건지령들은 조건을 만족하는가 그렇지 않은가에 따라 과제를 수행한다. if지령은 여기에서 제일 간단한 형태이다. if/else지령은 두가지 조건을 허용하며 if/elif/else지령은 다중조건을 허용한다. Bash는 두가지 조건형태를 검사할수 있다. 즉 지령이 성공인가 아니면 실패인가와 표현식이 참인가 아니면 거짓인가 하는것이다. 이 경우 항상 탈퇴상태가 사용된다. 탈퇴상태 0은 성공이나 참을 의미하고 령이 아닌 탈퇴상태는 실패나 거짓을 의미한다. 상태변수 ?는 탈퇴상태를 현시하는 수값을 가지고있다.

### 4.3.2. 내부지령 test

내부지령 test를 사용하여 표현식을 평가한다. 이 지령은 또한 중괄호 [ ] 로도 표현할수 있다. test지령 그자체를 사용할수도 있고 표현식을 한개 중괄호안에 넣을수도 있다. 쉘메타문자확장을 단순한 test지령으로 평가하는 식에서나 중괄호를 사용할 때에는 진행하지 않는다.

bash2.x판본이상에서는 2중중괄호[[]](내부합성지령 test)를 표현식을 평가하는데 사용할수 있다. 단어를 분리하여 변수에 저장하지 않고 패턴대조가 가능할뿐만아니라 메타문자에 대한 해석도 수행한다. 공백이 포함되여있는 자모문자렬은 인용부호안에 넣어야 하며 공백의 유무에 관계없이 문자렬을 패턴의 일부분으로가 아니라 정확한 문자렬로서 평가하려면 인용부호안에 넣어야 한다. test지령에서는 론리연산자 &&(and)와 ||(or)를 -a와 -o추가선택으로 교체한다.

test지령은 산수연산식을 평가할수 있으며 c에서와 같은 풍부한 연산자들을(bash 2.x) 써서 사용하려면 let지령어를 써야 한다. let지령을 2중소괄호안에 식을 넣어서 다르게 나타낼수 있다.

test지령, 합성지령, let지령을 사용하여도 검사된 식의 결과는 성공을 나타내는 상태 0과 실패를 나타내는 상태 0이 아닌 값을 가진다.(표 4-2)

표 4-2. test지령의 연산자

| test연산자 | 참인 경우 |
|---|---|
| 문자렬검사 | |
| [string 1 = string 2] | 문자렬 1이 문자렬 2와 같다.(=량쪽에 공백을 사용하여야 한다.) |
| [string 1 == string 2] | Bash 2.x에서 = 대신에 ==를 사용할수 있다. |
| [string 1 != string 2] | 문자렬 1이 문자렬 2와 같지 않다. |
| [문자렬] | 문자렬은 빈값이 아니다. |
| [-z string] | 문자렬의 길이가 0이다. |
| [-n string] | 문자렬의 길이가 0이 아니다. |
| [-l string] | 문자렬의 길이(문자의 개수) |
| 론리검사 | |
| [string 1 -a string 2] | 문자렬 1과 문자렬 2가 둘다 참이다. |
| [string 1 -o string 2] | 문자렬 1과 문자렬 2의 둘중 하나가 참이다. |
| [!string 1] | 문자렬 1과 같지 않다. |
| 론리검사(복합검사) | |
| [[pattern1 && pattern2]] | 패턴 1과 패턴 2가 둘다 참이다. |
| [[pattern1 \|\| pattern2]] | 패턴 1과 패턴 2의 둘중 하나가 참이다. |
| [[!pattern]] | 패턴과 같지 않다. |
| 옹근수검사 | |
| [integer1 -eq integer2] | 옹근수 1이 옹근수 2와 같다. |
| [integer1 -ne integer2] | 옹근수 1이 옹근수 2와 같지 않다. |
| [integer1 -gt integer2] | 옹근수 1이 옹근수 2보다 크다. |
| [integer1 -ge integer2] | 옹근수 1이 옹근수 2와 같거나 크다. |
| [integer1 -lt integer2] | 옹근수 1이 옹근수 2보다 작다. |
| [integer1 -le integer2] | 옹근수 1이 옹근수 2보다 작거나 같다. |
| 파일시험을 위한 2항연산자 | |
| [file 1 -nt file 2] | 만일 파일 1이 파일 2보다 새것이면 참이다. |
| [file 1 -ot file 2] | 만일 파일 1이 파일 2보다 낡은것이면 참이다. |
| [file 1 -ef file 2] | 만일 파일 1과 파일 2가 같은 장치이거나 또는 같은 inode번호를 가지면 참이다. |

표 4-3. let지령연산자

| 연 산 자 | 의 미 |
|---|---|
| -, + | 덜기와 더하기 |
| !~ | 부정 |
| *, /, % | 곱하기, 나누기, 나머지 |
| +, - | 더하기, 덜기 |
| bash 2.x이후에 추가된 let연산자 | |
| <<, >> | 왼쪽밀기, 오른쪽밀기 |
| <=, >=, <, > | 비교연산자들 |
| ==, != | 같기, 같지 않기 |
| & | 비트적 |
| ^ | 비트안맞음론리합 |
| \| | 비트론리합 |
| && | 론리적 |
| \|\| | 론리합 |
| ==, /=, %=, +=, -=, <<=, >>=, &=, ^=, \|= | 대입 및 복합연산자 |

### 4.3.3. if지령

어떤 지령이 일정한 조건하에서만 실행되도록 하는데 if지령을 리용한다. 만일 파일이 오유든가 혹은 실행에서 오유가 있을 때 오유통보문이 현시되도록 할수 있다.

간단한 if명령문구조는 다음과 같다.

- if 지령

```
if <조건>
then <지령>
fi
```

조건값은 다른 스크립트나 프로그람의 실행되돌이값이다. 되돌림값이 령이라면 조건은 false로 된다. 만일 되돌림값이 령이 아니면 조건은 true로 된다. 대부분의 경우에 령되돌이값은 성공을 가리키고 령아닌 되돌이값은 오유코드의 번호를 의미한다. fi는 if지령어의 끝을 의미한다. 조건구조에 쓰일수 있는 test라고 부르는 쉘내부지령이 있다.

실례로 지령이 본문파일의 내용을 현시하는 지령인데 만일 파일이 존재하지 않는다면 다음과 같이 할수 있다.

```
# !/bin/sh
if test -r $1
then
    cat $1
fi
```

여기서 test의 -r추가선택은 만일 파일이 존재하고 읽을수 있으면 령이 아닌값을 되돌리며 조건식은 참으로 된다.

test의 추가선택에 의하여 파일이 등록부인가 아니면 소케트인가, 또는 령길이의 파일인가, 쓰기가능한가 아니면 읽기가능한가 또는 특수한 사용자나 그룹에 의하여 소유되였는가를 결정할수 있다. test는 한 파일과 다른 파일을 비교할수 있고 수값과 문자들을 비교할수 있으며 또한 론리식 and와 or로 다른 조건과 결합할수 있다.

- if/else지령

기본문법은 다음과 같다.

```
if <조건>
then
    <지령>
else
    <지령>
fi
```

다음의 스크립트는 본문파일을 판본에 따라 정렬하는 스크립트이다.

```
# !/bin/sh
if ! test -r $1
then
    echo “The file $1 does not exit”
    exit 1
fi
if sort $1>temporary.work.file
then
    rm -f $1
    mv temporary.work.file $1
    echo “Done...”
else
    echo “Error.Unable to sort $1”
    exit 1
fi
exit 0
```

만일 파일이 존재하지 않는다면 통보문이 현시되며 령아닌 되돌이값을 가지고 탈퇴한다.

sort지령은 입구본문파일을 정렬하는데 쓰인다.

sort지령이 if문에서 실행되기때문에 sort의 탈퇴코드에 따라서 실행이 달라진다.

만일 어떤 원인에 의해서 sort가 성공하지 못한다면 령아닌 값을 되돌리며 오유통보문이 현시된다. 즉 오유가 있을 때 이 스크립트는 령아닌 값을 되돌리며 오유가 발생했다는것을 가리킨다. 만일 성공하면 원래 파일이 삭제되고 그 대신에 새로운 파일이 그 이름을 가진다.

- if/else/elif지령

if지령의 또 다른 형태도 있다. 그 형식은 다음과 같다.

```
if <조건>
then
    <지령>
elif <조건>
then
    <지령>
elif <조건>
then
    <지령>
....
elif <조건>
then
    <지령>
fi
```

만일 어떤 상품을 팔 때 사는 개수에 따라서 개당 가격이 다르다고 하면 입력한 개수에 따라서 개당 가격을 출력하는 프로그람을 작성하여야 한다.

즉 100개이상을 사면 290원이고 51개부터 100개까지를 사면 310원이며 11개부터 50개까지를 사면 345원 10개 이하이면 400원 한다고 하자.

이것을 스크립트로 작성하면 다음과 같다.

```
#!/bin/sh
if test $1 -gt 100
then
    price=290
elif  test $1 -gt 50
then
    price=310
elif  test $1 -gt 10
then
    price=345
else
    price=400
fi
echo $price
exit 0
```

test지령은 내부지령으로써 []를 쓴것과 동일하다. 즉 다음과 같이 쓸수 있다.

if [$1 -gt 100]은 if test $1 -gt 100과 동일하다.

### 4.3.4. 파일검사

사용자가 스크립트를 작성할 때 어떤 파일에 대한 정보가 필요한 경우가 있다.
즉 파일들이 적당한 허가가 설정되여있는가, 파일형식, 기본속성이 적당한지를 알아봐야 하는것이다. 이러한 작업은 스크립트를 정확히 작성하는데서 매우 중요하다.

표 4-4. 파일시험연산자

| 검사연산자 | 참일 때의 의미 |
|---|---|
| -b 파일이름 | 블로크파일 |
| -c 파일이름 | 문자파일 |
| -d 파일이름 | 등록부존재 |
| -e 파일이름 | 파일존재 |
| -f 파일이름 | 정규파일이 존재하지만 등록부는 아니다. |
| -G 파일이름 | 파일이 존재하면서 유효한 그룹을 가지고있다. |
| -g 파일이름 | setGID가 설정되여있다. |
| -k 파일이름 | sticky bit가 설정되여있다. |
| -L 파일이름 | 파일이 기호련결이다. |
| -P 파일이름 | 파일은 이름 붙인 파이프이다. |
| -O 파일이름 | 파일이 존재하며 유효한 사용자 ID를 가지고있다. |
| -r 파일이름 | 파일은 읽기가능하다. |
| -S 파일이름 | 파일은 소케트이다. |
| -s 파일이름 | 파일의 크기가 령이 아니다. |
| -tfd | fd(파일서술자)가 말단에서 열려져있다. |
| -u 파일이름 | setUID가 설정되여있다. |
| -w 파일이름 | 파일은 쓰기가능하다. |
| -x 파일이름 | 파일은 실행가능하다. |

## 제4절. 순환지령

순환지령들은 한개 지령이나 여러개의 지령들을 지정된 회수만큼 혹은 일정한 조건을 만족시킬 때까지 실행시키는데 리용한다. bash쉘에는 3가지 순환형태 즉 for순환, while순환, until순환이 있다.
while명령문은 조건표현식이 참일동안 순환한다. until명령문은 조건식이 거짓일동안 실행한다. for명령문은 목록의 매성원들을 한번 실행한다. break와 continue지령은 현재 순환에서 다른 순환에로 나올 때 쓸수 있다.

### 4.4.1. for지령

for순환지령은 지령을 항목에 있는 회수만큼 실행시키는데 사용한다. 실례로 파일들이나 사용자이름목록에 대하여 같은 지령들을 실행시키기 위해 순환지령을 사용할수 있다. for지령 다음에는 사용자정의변수가 놓이고 그 다음에 예약어 in, 단어들의 목록이 놓인다.

첫번째 순환에서는 단어목록에 있는 첫번째 단어가 변수에 대입되고 그 다음에 목록에서 밀려난다. 단어를 변수에 대입하면 순환본체에 들어가 do와 done예약어사이에 있는 지령들을 실행한다. 다음번 순환에서는 두번째 단어를 변수에 대입하며 우와 같은 과정들이 계속 반복된다. 순환본체는 do예약어로 시작하여 done예약어로 끝난다. 목록에 있는 모든 단어들이 밀려나면 순환은 끝나며 프로그람조종권은 done예약어 다음에로 넘어간다.

**형식**

```
for <변수이름> [ in <목록>]
do
<지령들>
done
```

**실례 4-5.**

(스크립트)

```
# !/bin/bash
# scriptname:forloop
1  for pal in Tom Dick Harry Joe
2  do
3      echo "Hi $pal"
4  done
5  echo "Out of loop"
```

(실행결과)

```
$ forloop
Hi Tom
Hi Dick
Hi Harry
Hi Joe
Out of loop
```

**설명**

1 for순환은 목록안에 있는 Tom, Dick, Harry, Joe를 하나씩 리용한 다음 밀기하면서 순환을 진행한다.(왼쪽에서부터 밀기하면서 그 값을 사용자정의변수 pal에 대입한다.) 모든 단어들이 밀기되여 단어목록이 비게 되면 순환은 끝나고 실행은 done예약어다음부터 시작된다. 첫번째 순환에서는 변수 pal에 단어 Tom을 대입하고 두

번째 순환에서는 Dick를, 그 다음 순환에서는 Harry를, 그 다음번 순환에서는 Joe를 대입한다.

2 do예약어는 목록다음에 놓이게 된다. 만약 같은 행에 쓰려면 반드시 반두점으로 구분해야 한다. 즉

for pal in Tom Dick Harry Joe; do

3 이 부분은 순환본체이다. Tom을 변수값으로 대입한 다음 순환본체에 있는 지령(즉 do와 done예약어사이의 모든 지령)들이 실행된다.

4 done예약어로 순환을 끝낸다. 단어목록의 마지막단어(Joe)가 대입된 다음 밀기되면 순환에서 벗어난다.

5 순환에서 벗어나 조종권이 넘어왔다.

**실례 4-6.**

(지령행)

```
1   $ cat mylist
    tom
    patty
    ann
    jake
```

(스크립트)

```
# !/bin/bash
# Scriptname:mailer
2   for person In $(cat mylist)
    do
3       mail $person < letter
        echo $person was sent a letter.
4   done
5   echo "The letter has been sent."
```

**설명**

1 mylist라고 하는 파일의 내용을 현시한다.

2 지령치환이 진행되고 mylist의 내용이 단어목록으로 된다. 첫 순환에서는 tom을 변수 person에 대입한 다음 한자리 밀기되여 patty로 교체되며 이와 같은 과정을 계속 반복한다.

3 순환본체에서는 매 사용자에게 letter파일을 우편으로 보낸다.

4 done예약어는 순환반복의 끝을 나타낸다.

5 목록에 있는 모든 사용자들에게 우편을 보낸 후 이 행을 실행한다.

**실례 4-7.**

(스크립트)

```
# !bin/bash
# Scriptname:backup
# Purpose:Create backup files and store
# them in a backup directory.
#
1   dir=/home/jody/kim/backupscript
2   for file In memo[1-5]
    do
3       if [ -f $file]
        then
            cp $file $dir/$file.bak
            echo "$file is backed up in $dir"
        fi
4   done
```

(실행결과)

```
memo1 is backed up in /home/jody/kim/backupscripts
memo2 is backed up in /home/jody/kim/backupscripts
memo3 is backed up in /home/jody/kim/backupscripts
memo4 is backed up in /home/jody/kim/backupscripts
memo5 is backed up in /home/jody/kim/backupscripts
```

**설명**

1 변수 dir에 여벌스크립트(backup script)가 기억되여있는 등록부가 대입된다.
2 단어목록은 memo로 시작하고 1~5까지의 수들로 끝나는 이름을 가진 현재작업등록부의 모든 파일로 이루어진다. 순환의 매 반복에서 변수 file에 모든 파일이름들을 한번에 하나씩 대입한다.
3 순환본체에 들어갈 때 파일의 존재성여부를 검사한다. 있으면 그 파일의 본래이름에 확장자 .bak를 붙여서 등록부 /home/jody/kim/backupscripts에 복사한다.
4 done은 순환의 끝을 나타낸다.

**실례 4-8.**

(스크립트)

```
# !/bin/bash
# Scriptname:greet
1   for name in $*             # same as for name In $@
2   do
```

```
        echo kim $name
3   done

(지령행)
$ greet cholsu inchol kangchol hyangmi
kim cholsu
kim inchol
kim kangchol
kim hyangmi
```

**설명**

1 $*와 $@는 모든 위치파라메터들의 목록으로서 이 경우에는 지령행에서 넘어온 인수들인 cholsu, inchol, kangchol과 hyangmi들로 확장된다. 목록에 있는 매 이름을 차례로 for순환에 있는 변수 name에 대입한다.
2 순환본체에 있는 지령들을 목록이 빌 때까지 실행한다.
3 done예약어는 순환본체의 끝을 나타낸다.

단어목록에 있는 $*와 $@변수는 $*와 $@가 확장될 때 인용부호안에 있지 않는한 그 기능들은 같다.

**실례 4-9.**

```
(스크립트)
# !/bin/bash
# Scriptname: permx
1  for file              # Empty wordlist
    do
2      if [ -f $file -a ! -x $file ]  # 혹은  if [[ -f $file && ! -x $file ]]
        then
3          chmod +x $file
       echo $file now has execute permission
       fi
   done

(지령행)
4  $ permx *
   adden now has execute permission
   checken now has execute permission
   doit now has execute permission
```

**설명**

1 만일 for순환에 단어목록이 주어지지 않으면 위치파라메터들을 반복한다. 이것은

for file $*와 같은 의미이다.
2 파일이름을 지령행에서 넘겨받는다. 쉘은 *를 현재작업등록부안의 모든 파일이름들로 전개한다. 파일이 보통 파일이고 실행허가가 없으면 3행을 실행한다.
3 처리한 매 파일에 실행허가를 추가한다.
4 지령행에서 쉘은 *를 통용기호로 해석하여 *대신에 현재등록부에 있는 모든 파일이름으로 교체한다. 이 파일이름들을 permx스크립트에 인수들로 넘긴다.

### 4.4.2. while지령

while지령은 뒤에 오는 식의 값을 평가하고 탈퇴상태가 0이면 순환본체에 있는 지령들을 실행한다. done예약어가 나타나면 조종은 순환의 시작부분으로 되돌아가고 while지령은 지령의 탈퇴상태를 다시 검사한다. while지령에 의해 평가되는 탈퇴상태가 0이 아닐 때까지 순환은 계속된다. 탈퇴상태가 0이 아니면 프로그람은 done예약어 다음부터 실행을 시작한다.

**형식**

```
while 지령
do
지령(들)
done
```

**실례 4-10.**

(스크립트)

```
# !/bin/bash
# Scriptname:num
1   num=0       # Initialize num
2   while (( $num < 10 ))       #or while [ num -lt 10 ]
    do
       echo -n "$num"
3      let num+=1               # Interement num
    done
4   echo -e "\nAfter loop exits, continue running here"
```

(실행결과)

```
  0 1 2 3 4 5 6 7 8 9
After loop exits, continue running here
```

**설명**

1 변수 num에 0을 대입하여 초기화를 진행한다.
2 num이 10보다 작으면 탈퇴상태가 0으로 되며 순환에 들어가게 된다.
3 순환본체에서 num값은 하나씩 증가된다. 만약 num값이 변하지 않는다면 순환은

무한히 반복되거나 프로쎄스가 끝날 때까지 계속된다.
4 echo -e지령에 의해 행바꾸기를 진행하고 문자렬을 현시한다.

**실례 4-11.**

(스크립트)

```
# !/bin/bash
# Scriptname:quiz
1   echo "누가 처음에 학교에 왔는가?"
    read answer
2  while [ "$answer" != "리철" ]
3  do
     echo  "틀렸습니다."
4     read answer
5  done
6  echo "옳습니다."
```

(실행결과)

```
$ quiz
누가 처음에 학교에 왔는가?김철
틀렸습니다.
누가 처음에 학교에 왔는가?박철
틀렸습니다.
누가 처음에 학교에 왔는가?리철
옳습니다.
```

**설명**

1 echo지령은 사용자에게 "누가 처음에 학교에 왔는가?"라는 재촉문을 현시한다. read지령은 사용자의 입력을 기다린다. 입력된 값은 변수 answer에 기억된다.
2 while순환이 시작되여 중괄호로 표시한 test지령이 식을 검사한다. 변수 answer가 《리철》과 같지 않으면 순환을 시작하여 do와 done사이에 있는 지령들을 실행한다.
3 do예약어는 순환본체의 시작이다.
4 사용자에게 재입력을 요구한다.
5 done예약어는 순환본체의 끝을 의미한다. 조종을 다시 while순환의 시작부분에로 되돌려보내고 식을 다시 검사한다. answer가 《리철》이 아니면 순환을 계속 반복한다. 사용자가 《리철》을 입력하면 순환이 끝나게 된다. 프로그람의 조종은 6행으로 넘어간다.
6 순환이 끝나면 여기에서 다음 지령이 실행된다.

**실례 4-12.**

(스크립트)

```
$ cat sayit
# !/bin/bash
# Scriptname:sayit
echo Type q to quit.
go=start
1  while [ -n "$go" ] # Make sure to double quote the variable
   do
2    echo -n I love motherland.
3    read word
4    if  [[ "$word" == [Qq] ]]
     then            # ["$word"=q-o "$word"=Q] Old style
        echo "I'll always love motherland!"
        go=
     fi
   done
```

(실행결과)

```
$ sayit
Type q to quit.
I love motherland.    ← When user presses Enter,the program continues
I love motherland.
I love motherland.
I love motherland.
I love motherland.q
I'll always love motherland!
```

**설명**

1 while다음에 있는 지령을 실행하고 그 탈퇴상태를 검사한다. test지령에 대한 n추가선택은 문자렬이 빈값이 아닌가를 검사한다. 변수 go는 초기값을 가지므로 검사가 성공되여 출력상태를 0으로 한다. 만일 변수 go가 2중인용부호안에 있지 않고 또한 그 변수가 빈값이라면 test지령은 다음과 같은 통보문을 내보낸다.
go: test: argument expected

2 순환을 시작한다. 문자렬 I love motherland.가 현시장치에 현시된다.

3 read지령은 사용자입력을 요구한다.

4 식을 평가한다. 사용자가 q나 Q를 입력한 경우 문자렬 I'll always motherland!가 현시되고 변수 go를 빈값으로 설정한다. while순환을 다시 시작할 때 변수가 빈값이므로 검사는 실패하며 순환은 끝난다. 조종은 done예약어 다음에로 이행한다. 이 실례에서 스크립트는 더 실행할 행이 없으므로 중단된다.

### 4.4.3. until지령

until지령은 while지령과 류사한데 다만 until다음의 지령이 실패일 때만 즉 지령이 탈퇴상태가 0이 아니면 순환이 진행된다. done예약어가 나타나면 조종은 순환의 시작부분에로 이행하며 until지령은 그 지령의 탈퇴상태를 다시 검사한다. until지령에 의해 평가되는 지령의 탈퇴상태가 0으로 될 때까지 순환은 계속된다. 탈퇴상태가 0으로 되면 순환에서 탈퇴되며 프로그람실행은 예약어 done다음부터 시작된다.

**형식**

```
until <조건식>
do
<지령들>
done
```

**실례 4-13.**

```
# !/bin/bash
1   until who | grep yongnam
2   do
        sleep 5
3   done
    talk yongnam @dragonwings
```

**설명**

1 until순환은 파이프에 있는 마지막지령인 grep의 탈퇴상태를 검사한다. who지령은 이 콤퓨터에 누가 가입하였는가를 표시하며 그 출력을 grep로 파이프처리한다. grep지령은 사용자 yongnam을 찾았을 때에만 0탈퇴상태를 되돌려준다.

2 만약 사용자 yongnam이 가입하지 않았다면 순환본체에 들어가 프로그람은 5s동안 정지상태에 놓인다.

3 yongnam이 가입하면 grep명령의 탈퇴상태는 0으로 되고 done예약어 다음의 명령이 실행된다.

**실례 4-14.**

(스크립트)

```
$ cat hour
# !/bin/bash
# scriptname:hour
1   let hour=0
2   until (( hour > 24 ))      # or [ $hour -gt 24 ]
    do
3       case "$hour" in
            [0-9]|1[0-1])    echo   "Good morning!" ;;
```

```
          12)      echo  "Lunch time." ;;
          1[3-7]) echo  "Siesta time." ;;
          *)       echo  "Good night." ;;
       esac
4   let hour+=1     # Don't forget to increment the hour
5   done

(실행결과)
$ hour
Good moring!
Good moring!
   …
Lunch Time.
Siesta time.
   …
Good night.
```

**설명**

1 변수 hour를 0으로 초기화한다.
2 test지령은 시간이 24보다 크거나 같은가를 비교한다. 만약 시간이 24보다 작으면 순환이 시작된다. 조건이 참일 때까지 순환은 계속 반복된다.
3 case명령은 변수 hour의 값을 검사하여 매 case명령문과 일치되는가를 검사한다.
4 hour변수는 let지령에 의해 증가된다.
5 done지령은 순환본체의 끝을 나타낸다.

### 4.4.4. select지령과 차림표

bash는 select지령을 차림표를 만드는데 리용한다. 수자들로 목록화된 항목들의 차림표를 표준오유로 현시하고 PS3재촉문을 현시한 다음 사용자의 입력을 기다린다. PS3의 기정값은 #?이다. 입력값은 차림표목록에 있는 수들중의 어느 하나가 되여야 한다. 입력은 쉘의 특수변수인 REPLY에 기억된다. REPLY변수에 있는 수자는 선택항목의 결정에 사용한다.

case지령은 select지령과 같이 리용되여 사용자가 차림표에서 한개를 선택하고 그에 기초하여 지령을 실행할수 있도록 한다. LINES와 COLUMNS변수들은 말단에 현시된 차림표항목들의 배치상태를 결정하는데 쓴다. 차림표가 표준오유로 현시될 때 그 매 항목의 앞에는 수자와 닫는 괄호가 있으며 PS3재촉문을 차림표의 맨 밑에 현시한다. select지령이 순환지령이기때문에 순환에서 벗어나기 위해서는 break지령을 쓰며 스크립트에서 탈퇴하기 위해서는 exit지령을 써야 한다.

**형식**

```
select var in wordlist
```

```
do
지령(들)
done
```

**실례 4-15.**

```
(스크립트)
# !/bin/bash
# Program name: runit
1   PS3="Select a program to execute: "
2   select program  in 'ls -F' pwd date
3   do
4       $program
5   done

(지령행)
$ runit
1) ls -F
2) pwd
3) date
select a program to execute: 2
/home/kim
1) ls -F
2) pwd
3) date
select a program to execute: 1
12abcrty abc12 doit* progs/ xyz
1) ls -F
2) pwd
3) date
select a program to execute: 3
Sun Mar 12 13:28:25 EST 2007
```

**설명**

1 PS3변수에 차림표목록의 맨 아래에 있는 재촉문을 지정한다.이 재촉문은 기정으로 $# 이며 재촉문을 출력한 후에 사용자의 입력을 기다린다.

2 select순환은 program이라는 변수와 차림표에 표시되는 세 단어 목록인 ls -F, pwd, date로 이루어진다. 이 목록에 있는 단어들은 LINUX지령들인데 임의의 다른 단어들 실례로 red, green, yellow, cheese, bread, milk, crackers 등도 사용할 수 있다. 만약 단어에 공백이 있으면 괄호로 묶어주어야 한다. 실례로 's -F'와 같이 한다.

3 do예약어는 순환본체의 시작을 의미한다.

4 사용자가 차림표에 있는 번호를 선택할 때 그 번호는 괄호가 붙은 번호다음에 있는 단어와 같다. 실례로 사용자가 번호 2를 선택하면 그것은 pwd로 되며 pwd는 변수 program에 기억된다. $program은 실행지령 pwd와 같다. 지령이 실행된다.
5 done예약어는 select순환에 있는 명령문들의 끝을 나타낸다. 조종은 순환의 시작부분에로 돌아간다. 이 순환은 사용자가 ^C를 누를 때까지 계속 실행된다.

**실례 4-16.**

(스크립트)

```
# !/bin/bash
# scropt name:goodboys
1  PS3="Please choose one of the three boys: "
2  select choice in tom dan guy
3  do
4     case $choice in
      tom)
         echo Tom is a student.
5        break;;              #break out of the select loop
6     dan | guy )
         echo Dan and Guy are workers.
         break;;
7     *)
         echo "$REPLY is not one of your choices" 1>&2
         echo "Try again." ;;
8    esac
9  done
```

(지령행)

```
$ goodboys
1)tom
2)dan
3)guy
Please choose one of the three boys: 2
Dan and Guy are workers.
$ goodboys
1) tom
2) dan
3) guy
Please choose one of the three boys: 4
4 is not one of your choices
```

```
Try again.
Please choose one of the three boys: 1
Tom is a student.
```

**설명**

1 두번째 행에 있는 select순환으로 만들어진 차림표아래에 PS3재촉문을 현시한다.
2 select순환에 들어간다. 목록에 있는 단어들이 차림표의 항목으로 현시된다.
3 순환본체가 시작된다.
4 목록에 있는 첫번째 값을 변수에 대입한 다음 목록에서 밀기를 진행하여 다음 항목이 첫번째 값으로 된다.
5 break명령문은 순환조종을 9행 다음에로 옮긴다.
6 dan 혹은 guy가 선택되면 다음에 있는 echo지령을 실행하며 그 다음에 break지령을 실행하여 조종을 9행 다음으로 보낸다.
7 내부 REPLY변수는 현재목록항목 즉 1, 2, 3과 같은 번호를 가지고있다.
8 case지령의 끝을 나타낸다.
9 done은 select순환의 끝을 나타낸다.

**실례 4-17.**

(스크립트)

```
# !/bin/bash
# Program name:ttype
# Purpose:set the terminal type
# Author:Andy Admin
1  COLUMNS=60
2  LINES=1
3  PS3="Please enter the terminal type: "
4  Select choice in wyse50 vt200 xterm sun
   do
5      case $REPLY in
           1)
6
               export TERM=$choice
               printf "TERM=$choice \n"
               break;;         # break out of the select loop
           2 | 3 )
               export TERM=$choice
               printf "TERM=$choice \n"
               break;;
           4)
```

```
              export TERM=$choice
              printf "TERM=$choice \n"
              break;;
7          *)
              echo -e "$REPLY is not a valid choice. Try again \n" 1>&2
8             REALY=  # Causes the menu to be redisplayed ;;
9         esac
10  done
```

(지령행)

```
$ ttype
1) wyse50    2) vt200    3) xtexm    4) sun
Please enter the terminal type : 4
TERM=sun
$ ttype
1) wyse50    2) vt200    3) xterm    4) sun
Please enter the terminal type : 3 TERM=xterm
$ ttype
1) wyse50    2) vt200    3) xterm    4) sun
Please enter the terminal type : 7
7 is not a valid choice. Try again.
1) wyse50    2) vt200    3) xterm   4) sun
Please enter the terminal type: 2
TERM=vt200
```

**설명**

1 COLUMNS변수는 차림표에서 말단의 너비를 설정한다. 기정값은 80이다.
2 LINES변수는 말단에서 select차림표의 행수를 조종한다. 기정값은 24행이다. LINES값을 1로 설정하면 차림표항목을 아래로 표시하는것이 아니라 한 행으로 표시한다.
3 PS3재촉문이 설정되여 차림표의 맨 아래에 나타난다.
4 select순환은 4개의 선택요소 즉 wyse50, vt200, xterm, sun으로 된 차림표를 현시한다. REPLY변수에 들어있는 사용자의 응답에 기초한 값들중의 하나를 변수 choice에 대입한다. 만일 REPLY가 1이면 choice에 wyse50을 대입하고 REPLY가 2이면 choice에 vt200을 대입하며 REPLY가 3이면 choice에 xterm을 대입하고 REPLY가 4이면 choice에 sun을 대입한다.
5 REPLY변수는 사용자가 선택한 항목으로 평가된다.
6 말단장치의 형태를 대입하고 반출하며 현시한다.
7 사용자가 1~4까지를 제외한 다른 값을 입력했을 때는 다시 입력할것을 요구한다. 그러나 차림표는 나타나지 않고 PS3재촉문만 나타난다.

8 REPLY가 빈값으로 설정되면 차림표를 다시 현시한다.
9 Select순환의 끝이다.

### 4.4.5. 순환조종

어떤 조건이 발생하면 순환에서 벗어나 순환의 시작부분에로 돌아가거나 무한순환을 중지할 필요가 있다. bash쉘에는 이와 같은 처리를 위한 순환조종지령들을 가지고있다.

- shift지령

shift지령은 지적된 회수만큼 파라메터목록을 왼쪽으로 밀기한다.

shift지령에 인수가 없으면 파라메터목록을 왼쪽으로 한번만 밀기한다. 목록을 밀기하면 밀기된 파라메터는 완전히 삭제된다. shift지령은 while순환에서 자주 쓰인다.

**형식**

```
shift [n]
```

**실례 4-18.**

(스크립트: 순환이 없는 실례)

```
# !/bin/bash
# Scriptname:shifter
1   set joe mary tom sam
2   shift
3   echo $*
4   set $(date)
5   echo $*
6   shift 5
7   echo $*
8   shift 2
```

(실행결과)

```
3  mary tom sam
5  Thu Mar 15 10:00:12 PST 2007
7  2007
8  shift:shift count must be=<$#
```

**설명**

1 set지령은 위치파라메터들을 설정한다. $1에 joe를, $2에 mary를, $3에 tom을, $4에는 sam을 대입한다. $*는 모든 파라메터들을 나타낸다.
2 shift지령은 파라메터를 왼쪽으로 밀기한다. 즉 joe가 밀기된다.
3 파라메터목록을 밀기한 다음에 현시한다.
4 set지령은 Linux의 date지령의 출력으로 위치파라메터를 재설정한다.

5 새로운 파라메터목록이 현시된다.
6 이때 목록을 왼쪽으로 5번 밀기한다.
7 새로운 파라메터목록을 현시한다.
8 주어진 파라메터개수이상으로 밀기하면 쉘은 표준오유통보를 내보내여 shift지령이 초과되는 파라메터들을 밀기할수 없다는것을 알려준다. $#는 위치파라메터의 총개수이다.

**실례 4-19.**

(스크립트:순환이 있는 실례)

```
# !bin/bash
# 이름:doit
# 목적:지령행인수를 리용한 밀기
1   while (( $# > 0 ))          #  혹은  [ $# -gt 0]
    do
2       echo $*
3       shift
4   done
```

(지령행)

```
$ doit a b c d e
a b c d e
b c d e
c d e
d e
e
```

**설명**

1 let지령은 수식을 검사한다. 위치파라메터들의 개수($#)가 0보다 크면 순환본체에 들어간다. 위치파라메터들은 지령행에서 인수로서 넘어온다. 인수는 5개이다.
2 모든 위치파라메터들을 현시한다.
3 파라메터목록을 왼쪽으로 한번 밀기한다.
4 이 행은 순환본체의 끝이므로 조종을 순환부의 시작부분에로 돌려보낸다. 매 순환에 들어갈 때마다 shift지령은 파라메터목록을 하나씩 감소시킨다. 첫번째 밀기를 진행한 다음 $#(위치파라메터들의 수)는 4개로 된다. $#가 0으로 되면 순환은 끝나게 된다.

**실례 4-20.**

(스크립트)

```
# !/bin/bash
# Scriptname:dater
```

```
# Purpose:set positional parameters with the set command
# and shift through the parameters.
1   set $(date)
2   while (( $# > 0 ))    #  or  [ $# -gt 0 ] Old style
    do
3       echo $1
4       shift
    done

(실행결과)
$ dater
Thu
Mar
15
19:25:00
EST
2007
```

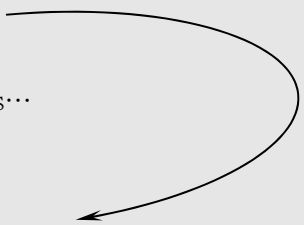
**설명**

1 set지령은 date지령을 출력하여 그것을 위치파라메터 $1~$6에 대입한다.
2 while지령은 위치파라메터들의 개수($#)가 0보다 큰가 작은가를 검사하고 크다면 순환본체에 들어가게 한다.
3 echo지령은 첫번째 위치파라메터인 $1의 값을 현시한다.
4 shift지령은 파라메터목록을 왼쪽으로 한번 밀기한다. 매 순환은 목록이 빌 때까지 밀기한다. 그때 $#가 0이 되면 순환이 끝나게 된다.

- break지령

내부 break지령은 순환에서 즉시 탈퇴할 때 쓴다. break지령이 실행된 다음 조종은 done예약어 다음에로 옮겨진다. break지령에 의해 제일 안쪽 순환에서부터 탈퇴되므로 겹쌓인 순환에서는 순환번호를 리용하여 순환의 탈퇴를 진행한다. 실례로 3개의 겹쌓인 순환이 있을 때 제일 바깥쪽 순환은 순환번호 3, 그다음 안쪽은 순환번호 2, 제일 안쪽은 순환번호 1이다. break지령은 무한순환에서 탈퇴할 때 효과적이다.

**형식**

```
break [n]
```

**실례 4-21.**

```
# !/bin/bash
# Scriptname:loopbreak
1   while true; do
2       echo Are you ready to move on\?
```

```
            read answer
3       if  [[ "$answer" == [Yy] ]]
            then
4           break
5       else
                ...commands...
            fi
6   done
7   printf "Here we are"
```

**설명**

1 true지령은 항상 상태 0을 가지고 탈퇴하는 Linux지령이다. 이것은 무한순환을 만드는데 자주 쓰인다. do명령문을 while지령과 같은 행에 놓을 때에는 반두점으로 구분해준다.
2 사용자에게 입력을 요구한다. 사용자입력을 변수 answer에 대입한다.
3 만일 $answer가 Y 또는 y이면 조종은 4행으로 이행한다.
4 break지령을 실행하면 순환에서 탈퇴하며 조종은 7행으로 이행하여 Here we are를 현시한다. 사용자로부터 Y 또는 y입력을 받을 때까지 프로그람은 입력을 계속 요구한다.
5 3행에서 검사가 실패하면 else지령을 실행한다. 순환본체가 done예약어에서 끝나면 조종은 1행에 있는 while지령에서부터 다시 시작한다.
6 순환본체의 끝이다.
7 break지령이 실행된 다음 조종은 이 행에서 시작한다.

- continue지령

continue지령은 지정한 조건을 만족하면 순환의 시작부분에로 조종을 돌려보낸다. 즉 continue지령다음에 있는 모든 지령들이 무시된다. 만일 겹쌓인 순환이라면 continue지령은 제일 안쪽 순환에 조종을 돌려 보낸다. 인수가 수로 주어졌다면 조종은 임의의 순환의 시작부분에서 시작할수 있다. 가령 3개의 순환이 겹쌓여있다면 제일 바깥순환은 순환번호 3, 다음 안쪽순환은 순환번호 2, 제일 안쪽순환은 순환번호 1이다.

**형식**

```
continue [n]
```

**실례 4-22.**

```
$ cat mail_list
ernie
john
richard
mealnie
greg
```

```
robin

(스크립트)
# !/bin/bash
# Scriptname:mailem
# purpose: To send a list
1   for name in $(cat mail_list)
        do
2       if [[$name==richard]]; then
3           continue
        else
4           mail $name < memo
        fi
5   done
```

**설명**

1 지령을 $(cat mail_list) 또는 'cat mail_list'로 대입한 다음 for순환은 mail_list파일에 있는 목록의 이름들을 반복한다.
2 이름이 richard로 되는 경우 continue지령이 실행되며 조종은 순환식을 평가하는 순환의 시작부분(1행)에로 되돌아간다. richard는 이미 목록에서 제외되였기때문에 다음 사용자 mealnie가 변수 name에 대입된다.
3 continue지령은 순환본체에 있는 나머지지령들을 뛰여넘어 조종을 순환의 시작부분에로 돌려보낸다.
4 richard를 제외하고 목록에 있는 모든 사용자들에게 memo파일을 우편으로 보낸다.
5 순환본체의 끝이다.

**- 겹쌓인 순환과 순환조종**

겹쌓인 순환을 진행할 때 break와 continue지령은 옹근수값을 인수로 하여 조종이 안쪽 순환에서 바깥쪽 순환에로 넘어갈수 있게 한다.

**실례 4-23.**

```
(스크립트)
# !/bin/bash
# Scriptname: months
1   for month in Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
        do
2       for week in  1 2 3 4
            do
                echo -n "Processing the month of $month. OK? "
                read ans
```

```
3         if [ "$ans" = n  -o -z "$ans" ]
              then
4             continue 2
              else
                 echo -n "Process week $week of $month? "
                 read ans
                 if [ "$ans" = n -o -z "$ans"  ]
                 then
5                continue
                 else
                    echo "Now processing week $week of $month."
                    sleep 1
                    # Commands go here
                    echo  "Done processing..."
                 fi
              fi
6       done
7   done
```

(실행결과)

```
$ months
Procossing the month of Jan. OK?
Procossing the month of Feb. OK? Y
Process week 1 of Feb? y
Now processing week 1 of Feb.
Done processing…
Procossing the month of Feb. OK? Y
Process week 2 of Feb? y
Now processing week 2 of Feb.
Done processing…
Procossing the month of Feb. OK? N
Procossing the month of Mar. OK? N
Procossing the month of Apr. OK? N
Procossing the month of May. OK? N
```

**설명**

1 바깥쪽 for순환이 시작된다. 이 순환에서 먼저 month에 Jan을 대입한다.
2 안쪽 for순환을 시작한다. 이 순환에서는 먼저 week에 1을 대입한다. 안쪽 순환을 다 끝내야 바깥쪽 순환으로 갈수 있다.
3 사용자가 n 또는 Enter건을 누르면 4행이 실행된다.

4 인수 2를 가진 continue지령은 두번째 바깥쪽 순환의 시작부분에서 조종을 시작한다. 즉 1행으로 이행한다. 인수가 없는 continue지령은 제일 안쪽 순환의 시작부분에로 조종을 돌려보낸다.
5 조종은 제일 안쪽 for순환에로 돌아간다.
6 done은 제일 안쪽 순환을 끝낸다.
7 done은 제일 바깥쪽 순환을 끝낸다.

### 4.4.6. I/O방향바꾸기 및 자식쉘

파이프 혹은 방향바꾸기를 리용하면 파일로부터 순환에로 입력을 바꿀수 있다. 출력도 마찬가지이다. 쉘은 자식쉘을 기동하여 I/O방향바꾸기와 파이프를 조종한다. 순환안에서 정의한 변수들은 순환이 끝나면 스크립트뒤부분에서 참고할수 없다.

**- 순환출력을 파일로 방향바꾸기**

bash순환에서의 출력을 현시장치로가 아니라 파일로 보낼수 있다. 이것을 실례 4-24에 보여주었다.

**실례 4-24.**

(지령행)

```
1  $ cat memo
    abc
    def
    ghi
    (스크립트)
    # !/bin/bash
    # Program name: numberit
    # Put line numbers all lines of memo
2  if let $(($# < 1 ))
   then
3      echo "Usage: $0 filename " >&2
       exit 1
   fi
4  count=1                      # Initialize count
5  cat $1 | while read line
   # Input is coming from file provided at command line
   do
6      let $((count == 1)) && echo "Processing file $1" > /dev/tty
7      echo -e "$count\t$line"
8      let count+=1
9  done > tmp$$                 # Output is going to a temporary file
```

```
10 mv tmp$$ $1

(지령행)
11 $ numberit memo
   'Processing file memo'
12 $ cat memo
   1 abc
   2 def
   3 ghi
```

**설명**

1 memo파일의 내용을 현시한다.
2 사용자가 스크립트를 실행하고있을 때 지령행에 인수를 주지 않으면 인수의 개수($#)는 1보다 작아지며 오유통보가 나타난다.
3 인수의 개수가 1보다 작으면 오유통보문을 stderr(>&2)에로 보낸다.
4 count변수에 1이 대입된다.
5 Linux의 cat지령은 $1에 보존된 파일이름의 내용을 현시하며 출력은 while순환으로 파이프된다. read지령의 첫번째 순환에는 파일의 첫 행이, 다음번 순환에는 파일의 두번째 행이 대입되며 이와 같은 과정이 계속된다. read지령은 입력이 성공하면 0을, 실패하면 1을 돌려준다.
6 count의 값이 1이면 echo지령을 실행하여 출력을 현시장치 /etc/tty에로 보낸다.
7 echo지령은 count값을 현시한 다음 파일에 있는 행을 현시한다.
8 count를 1만큼 증가시킨다.
9 전체 순환의 출력 즉 $1에 있는 파일의 첫번째 행을 제외한 매 행을 파일 tmp$$로 방향바꾸기한다. 첫번째 행은 말단 /dev/tty에로 방향바꾸기한다.
10 tmp파일은 $1로 지정된 파일이름으로 변경된다.
11 프로그람이 실행된다. 처리하려는 파일이름은 memo이다.
12 스크립트가 끝난 후 memo파일의 매 행번호와 함께 내용이 현시된다.

**- 순환출력을 Linux지령으로 파이프하기**

출력은 파이프를 리용하여 다른 지령의 입력으로 사용될수 있으며 방향바꾸기를 리용하여 파일로 보낼수도 있다.

**실례 4-25.**

```
(스크립트)
# !/bin/bash
1  for i in 7 9 2 3 4 5
2  do
       echo $i
3  done | sort -n
```

(실행결과)

```
2
3
4
5
7
9
```

**설명**

1 정렬되지 않은 수들의 목록에 대하여 for순환을 진행한다.
2 순환본체에서 수자들을 출력한다. 이 출력을 수값정렬인 sort지령의 입력으로 한다.
3 done예약어다음에 파이프를 생성한다. 순환은 자식쉘에서 실행된다.

**- 배경에서의 순환실행**

순환을 배경에서 실행시킬수 있다. 프로그람은 순환이 끝나기를 기다리지 않고 계속 실행될수 있다.

**실례 4-26.**

(스크립트)

```
# !/bin/bash
1  for person in bob jim joe sam
   do
2      mail $person < memo
3  done &
```

**설명**

1 for순환은 단어목록에 있는 매 이름 bob, jim, joe, sam들을 하나씩 처리한다. 매 이름들이 차례로 변수 person에 대입된다.
2 순환본체에서 매 사람들에게 memo파일의 내용을 우편으로 보낸다.
3 done예약어의 끝에 있는 &기호는 순환을 배경방식에서 실행하게 한다. 프로그람은 순환이 실행될동안 계속 집행된다.

### 4.4.7. IFS와 순환

쉘의 내부마당분리기호(IFS)에는 공백, 타브, 행바꾸기가 있다. 이것은 read, set, for와 같이 단어목록을 해석하는 지령에서 단어(어휘)분리기호로서 쓰인다. 가령 목록에서 서로 다른 분리기호를 쓰려면 사용자가 재설정해주어야 한다. 값을 변화시키기 전에 다른 변수에 IFS의 원래 값을 기억시키는것이 좋다. 그렇게 하면 필요에 따라 그의 기정값으로 돌아가기가 쉽다.

실례 4-27.

(스크립트)

```
$ cat runit2
# /bin/bash
# Script is called runit.
# IFS is the internal field separator and defaults to
# spaces, tabs, and newlines.
# In this script it is changed to a colon.
1  names=Tom:Dick:Harry:John
2  oldifs="$IFS"  # save the original value of IFS
3  IFS=":"
4  for persons in $names
   do
5     echo  Hi $persons
   done
6  IFS="$oldifs"            # reset the IFS to old value
7  set Jill Jane Jolene     # set positional parameters
8  for girl in $*
   do
9     echo Howdy $girl
   done
```

(실행결과)

```
$ runit2
5  Hi Tom
   Hi Dick
   Hi Harry
   Hi John
9  Howdy Jill
   Howdy Jane
   Howdy Jolene
```

설명

1 변수 names에 문자렬 Tom:Dick:Harry:John을 저장한다. 매 단어들은 두점으로 분리되여있다.

2 IFS의 원래 값을 다른 변수 oldifs에 대입한다. IFS의 값에 공백이 있으므로 인용부호안에 넣어 기억한다.

3 IFS에 두점을 대입한다. 두점은 단어들을 분리시키는데 쓰인다.

4 변수를 대입한 다음 두점을 단어들사이의 내부마당분리기호로서 사용하여 매 이름들에 대해 for순환을 진행한다.

5 단어목록의 매 이름들이 현시된다.
6 IFS는 oldifs에 기억되였던 원래 값으로 다시 대입된다.
7 위치파라메터들을 설정한다. $1에 Jill을, $2에 Jane을, $3에는 Jolene을 대입한다.
8 $*는 모든 위치파라메터들인 Jill, Jane, Jolene과 같다. for순환은 순환이 반복될 때마다 매 이름들을 girl변수에 대입한다.
9 파라메터목록에 있는 매 이름들이 현시된다.

## 제5절. 정규표현식

정규표현식은 문자규칙을 배렬한 문자렬이다. 문자규칙은 문자를 결정하는데 어떤것이 정합되고 어떤것이 정합되지 않는가 등을 적용하는 규칙이다.

다음으로 규칙들의 내용을 설명하자.

정규표현식에는 특수한 의미가 있는 일부 문자들이 있다.

실례로 별표(*)는 하나의 수자 또는 어떤 문자렬을 가리키는 통용문자이다. 물음기호(?)는 하나의 문자를 가리키는 통용문자이며 괄호([) 와 (])는 문자렬의 그룹범위를 지정하는데 쓰인다. 수직띠(|)는 두 정규표현식사이의 론리합을 의미한다. 다음의 표에 정규표현식과 그에 정합되는 문자들을 보여주었다.

표 4-5. 정규표현식에 정합되는 실례들

| 표현식 | 정합되는 문자들 |
|---|---|
| *a* | atom, bat, amazon, extra |
| *a | extra, area, medusa |
| a* | apple, amount |
| *oo* | loop, shoot, bookkeeper, zoo |
| m?x | mix, max, mox |
| m?? | mad, mrs, mix |
| ?amp | camp, ramp, samp |
| [abcd] | a, b, c, d |
| [crd]amp | camp, ramp, damp |
| [b-d]ad | bad, cad, dad |
| [a-z][0-9] | a1, b4, z9 |
| hm[13579] | hm1, hm5, hm9 |
| C[o0][bB] | Cob, C0B, CoB |
| h*n\|c*t | hitman, cement |

## 제6절. 스크립트와 프로그람의 실행

### 4.6.1 쉘실행추가선택

bash지령을 리용하여 쉘을 실행할 때 동작방식을 변경시키기 위한 추가선택들을 가질수 있다. 두가지 형태의 추가선택들 즉 단일문자추가선택과 다중문자추가선택들이 있다. 단일문자추가선택은 한개의 단일문자와 한개의 가로선으로 되여있다. 다중문자추가선택은 2개의 가로선과 여러개의 물음표들로 이루어진다. 다중문자추가선택들은 단일문자추가선택보다 앞에 나와야 한다.

대화형등록가입쉘은 보통 -i(대화형쉘을 기동한다.), -s(표준입력으로부터 읽는다.) -m(일감조종을 할수 있게 한다.)으로 시작한다.

표 4-6. bash2.x 쉘실행 추가선택

| 추가선택 | 의 미 |
|---|---|
| -c string | string을 읽어서 지령을 실행한다. |
| -I | 대화형방식으로 쉘이 기동한다. TERM, QUIT, INTERRUPUT는 무시된다. |
| -s | 지령들이 표준입력으로부터 읽어지고 위치파라메터들이 설정되게 한다. |
| -r | 제한된 쉘을 시동한다. |
| -- | 추가선택들의 끝을 알리고 추가선택처리가 더이상 진행되지 않게 한다. - 또는 -뒤에 있는 임의의 인수들은 파일이름과 인수들로 취급된다. |
| -dump_strings | -D와 같다. |
| --help | 내부지령에 대한 사용법통보문을 현시하고 탈퇴한다. |
| --login | bash가 등록가입쉘로 되게 한다. |
| -noediting | bash가 대화형으로 시동할 때 readline서고를 리용하지 않는다. |
| -noprofile | bash쉘은 시동할 때 초기화파일들인 /etc/profile, ~bash_profile 또는 ~/.bash_login, ~/.profile 을 읽지 않는다. |
| -norc | bash는 대화형쉘들에 대하여 ~/.bashrc파일을 읽지 않는다. 쉘을 sh로 기동한다면 기정값은 on이다. |
| -posix | bash의 동작방식이 표준 POSIX 1003.2가 아니면 그에 맞도록 변경시킨다. |
| -quiet | 쉘이 기동할 때 어떤 정보도 현시하지 않는다. |
| -refile파일 | Bash가 대화형이면 ~/.bashrc대신 파일을 초기화파일로 읽는다. |
| -restricted | 제한된 쉘을 시작한다. |
| -version | bash쉘에 대한 판본정보를 현시한다. |

스크립트가 다른 스크립트를 실행하는 방법에는 여러가지가 있다. 어떤 스크립트가

다른것을 실행시킬 때 새롭게 실행되는것을 자식스크립트라고 하며 본래 스크립트를 부모스크립트라고 부른다. 부모쉘은 자식쉘이 끝날 때까지 대기한다. 두개의 부모와 자식쉘이 동시에 실행될수 있으며 부모쉘은 자식쉘이 끝날 때까지 대기한다. 부모는 실행을 중지할수 있으며 이때 조종권을 자식쉘에 넘기며 자식은 부모의 위치에 놓이게 된다.

### 4.6.2. 단순실행

이것은 한 스크립트에 의해 다른 스크립트가 실행되는 가장 일반적인 방법이다. 단순히 실행하려는 스크립트를 지령행에 입력하면 된다. 부모스크립트는 자식스크립트가 계속하여 실행을 끝낼 때까지 중지하고 대기한다.

조건표현식에서 자식프로쎄스는 조건식이 령 혹은 령아닌값으로 될 때 탈퇴한다.

실례로 다음의 스크립트는 showstatus의 값이 령이든가 혹은 령이 아닌가에 관계없이 탈퇴하여 현시한다.

```
# !/bin/sh
if [ showstatus ]
then
    echo "True is represented by non-zero value"
else
    echo "False is represented by zero"
fi
exit 0
```

### 4.6.3. 자식프로쎄스에서의 탈퇴

하려는 작업의 마지막에 &기호를 입력하면 그 작업은 배경에서 진행된다. 이것은 지령행에서 실행할수 있는 모든것을 내부에서 실행되도록 한다. 다음의 코드는 지령을 찾아 실행하고 그 결과를 다른 파일로 출력한다.

```
find / -name "Xt*.c" -print > xtlist&
```

&기호로 지령이 끝났기때문에 실행은 배경에서 진행된다. 부모프로쎄스는 다른것을 할수 있도록 자유로워진다.

실제로 부모프로쎄스는 find지령의 프로쎄스번호로 시작할것이며 그것들은 거의 동시에 끝날것이다. 대기지령은 모든 자식프로쎄스가 완성될 때까지 부모프로쎄스를 중지시키는데 사용할수 있다.

```
#!/bin/sh
find / -name "Xt*.c" -print > xtlist&
find / -name "fred" -print > fredlist&
echo waiting ...
wait
echo Done...
exit 0
```

이 실례에서 두개의 find지령이 실행되는데 스크립트는 대기지령상에서 계속된다. wait지령은 두개의 자식프로쎄스가 완성될 때까지 대기하며 스크립트가 실행을 계속할 때 파일 xtlist와 fredlist는 자료로 채워지게 된다. 내장지령 wait를 리용하면 특수한 자식프로쎄스가 완성될 때까지 기다릴수 있다. 매개의 새로운 프로쎄스는 체계에 의하여 할당된 유일한 ID번호를 가진다. 이 번호는 프로쎄스가 실행중인지 아닌지를 판단하는데 쓰일수 있다. 다음의 실례는 두개의 자식프로쎄스가 시작하지만 둘중 하나의 프로쎄스가 끝날 때까지 대기한다.

```
# !/bin/sh
find / -name "Xt*.c" -print > xtlist &
IdNumber=$!
find / -name "fred" -print > fredlist &
echo Waiting for $IdNumber
wait $IdNumber
echo Done...
exit 0
```

특수한 기호인 $!는 가장 최근에 시작한 배경프로쎄스의 ID번호이다.
이 번호는 wait지령을 지원하며 쉘은 하나의 자식프로쎄스만을 대기한다.

### 4.6.4. 현재 프로쎄스의 교체

현재 프로쎄스를 즉시 중지시키고 그대신에 다른 프로쎄스를 기동시키는 특수지령이 있다. 다음에 그 실례를 보여주었다.

```
exec unroll
```

이 코드행은 현재의 스크립트를 즉시 중지시키고 기억기에 unroll이라는 스크립트를 불러들이고 실행한다.

우와 같은 결과는 배경에서 자식프로쎄스를 실행하고 부모프로쎄스에서 탈퇴할 때에도 얻을수 있다. exec지령을 리용하면 새로운 프로쎄스가 기동되지 않기때문에 더 효과적이다. 이것은 변경된 프로그람을 즉시 실행시키는데 효과적이다.

## 제7절. 스크립트를 유용하게 쓸수 있는 지령

Linux에 제공되는 지령은 수백개가 된다. 이 많은 지령들은 수년간에 걸쳐 발전되여 왔으며 그들 매개는 어느것이나 어디에인가에 필요한것이다. 그것들중 많은것은 문제를 정확히 풀도록 하는데 쓰인다.

그것들중에는 스크립트에서 매우 자주 쓰이는 지령들도 있다.

여기서는 스크립트에서 자주 쓰이는 지령들에 대하여 보기로 한다.

### 4.7.1. awk

Linux는 GNU를 구현한 POSIX표준의 awk프로그람언어를 가지고있다. 쉘프로그

람작성에서 아주 유용하게 사용할수 있는 구문이 awk구문이다. AWK라는 이름은 개발자들의 이름에서 첫글자를 모아서 만들어졌다. 개발자들은 Aho, Weinberger 그리고 Kemigham이였는데 이들은 패턴조작 및 연산을 효과적으로 수행할 도구를 만들기 위해 노력했고 그 결과물이 바로 AWK이다. AWK는 지령수행결과나 파일안의 자료내용을 한행씩 읽어들이는 기능과 행안의 매개 단어들을 끊어내고 조작 및 연산할수 있는 기능 등을 제공하고있다.

이러한 기능들로 하여 일반자료들을 원하는 형태로 가공하여 정보화실현에서 많은 도움을 받을수 있다. 많은 프로그람작성자들이 AWK를 리용하여 쉘프로그람을 작성하는 리유가 바로 여기에 있다. 그러면 AWK의 문법형식부터 살펴보도록 하자.

파일에서 자료를 얻어오려면

```
awk '패턴 {실행문} 패턴 {실행문} …패턴 {실행문} '파일명
```

지령수행결과에서 원하는 자료를 얻어내려면 다음과 같이 한다.

지령수행: awk '패턴 {실행문} 패턴 {실행문} ...패턴 {실행문} '

만일 련속된 "패턴 {실행문} "구문을 여러 행으로 바꾸려면 다음과 같이 한다.

```
패턴 {실행문} \
패턴 {실행문} \
패턴 {실행문} ...
```

awk에서 사용할 구문을 파일안에 작성한 다음 awk에서 해당 파일을 사용할수도 있다. 이런 경우에는 -f추가선택을 준 뒤 지령어가 작성된 파일을 입력해주면 된다. 이 기능을 활용하면 마치 함수를 호출해서 사용하는것 같은 효과를 얻을수 있다. 즉 많이 사용될 내용을 파일로 미리 작성해둔 다음 필요할 때 해당 파일을 불여서 사용하면 된다.

다음은 awk를 리용한 실례이다.

```
- DiskUsageCheck.sh-
# !/bin/sh
usage=`df -k $1 | /bin/awk '{rem = 0} {n += 1} {a = $3} {b= $4} \
n== 2 { rem = int(a/(a+b) * 100) ; print rem } \
END{ }'`
echo $usage
if [ $usage -ge 90 ]
then
     echo "DISK($usage) - 위험한 상태"
elif [ $usage -ge 70 ]
then
     echo "DISK($usage) - 주의가 필요한 상태"
else
```

```
        echo "DISK($usage) - 보통 상태"
    fi
```

우의 프로그람을 보면 df -k $1지령을 수행하고 결과로 나온 통보문을 awk지령에 넘겼다. 그리고 awk지령어가 그 통보문을 리용하여 각종 계산을 수행한 후에 최종 결과값을 usage라는 변수에 할당하였다. 그 다음 if문에서 usage에 할당된 값에 따라 화면에 출력하고있다.

여기서 주목하는 부분은 awk가 사용된 문장이다. 먼저 간단히 우의 문장을 설명한 후 awk지령어의 사용법에 대해 자세히 알아보도록 하자.

례제를 보면 awk를 사용하면서 먼저 rem이라는 변수를 0으로 초기화한것을 볼수 있다. 그다음 df -k에서 출력된 문장을 한 행씩 읽으면서 3번째 단어와 4번째 단어를 각각 a와 b에 대입한다. 그리고 읽어들인 문장이 두번째 문장이면 디스크사용량을 프로수로 계산하기 위해 int(a/(a+b)*100)이라는 연산을 진행한다. 연산이 끝나면 그 결과값을 rem변수에 할당하고 현시한다.

다음 실례를 또 하나 들어보자. 례를 들어 다음의 문장으로 된 'funcl'이라는 파일을 만들었다고 하자.

{print $1}

그리고 다음과 같이 실행한다.

awk -f funcl diskUsage.sh

그러면 diskUsage.sh파일속에 있는 매 행의 문장들중 첫번째 단어들이 화면에 출력된다. 패턴에 들어갈수 있는 예약어로는 "BEGIN"과 "END" 그리고 "각종 수식"이 들어갈수 있다. 이때 "BEGIN"은 자료를 읽기전에 실행하는 문장을 정의할 때 사용한다. "END"예약어는 awk가 모든 자료를 읽은 후에 실행하는 문장을 정의할 때 사용된다. 그리고 "각종수식"의 경우 일반 조건문처럼 수식을 실행한 후 결과가 령(0)이나 빈값(Null)이 아닐 때 이어서 나오는 문장을 실행하게 된다.

수식이나 실행문에는 C언어 등에서 제공하는 각종 연산자를 대부분 활용할수 있다. 례를 들어, +=, -=, *=, ++,-, ||, &&, ==, <=, != 등의 연산자도 사용할수 있다. 그리고 awk내부에서도 조건문과 반복문을 그대로 적용할수 있다. 이러한 기능들때문에 awk자체를 하나의 작은 개발도구로 인식하기도 한다.

AWK문을 활용하면서 내부에 자동할당되여있는 변수를 활용할수도 있다. 이러한 변수에는 입력된 파일의 이름을 가지고있는 "FILENAME", 현재 행의 번호를 가지고있는 "NR", 현재 행의 단어개수를 가지고있는 "NF", 입력단어의 분리자인 "FS"와 출력단어의 분리자인 "OFS" 등이 있다.

그러면 지금까지 소개한 예약어와 변수를 활용한 례제를 보도록 하자. 아래의 례제는 종업원정보가 입력된 파일을 읽은 후 출력하려고 하는 양식에 맞게 내용을 변경하고 총 입력된 종업원의 수가 얼마인지 표시하는 프로그람이다.

```
- awkTest.sh -
# !/bin/sh
/bin/awk -F: \
'BEGIN { FS= "|"; print "=== 주소록 출력 ===" ; print ""}\
  { n +=1 } { name = $1 } { telNo = $2 } { addr = $3 }\
  { print "<" name ">" } \
```

```
    { print "전화번호 : " telNo}\
    { print "주소 : " addr; print "" };\
  END { print "총 "n"명의 주소가 수록되였습니다." }'\
   address.txt
```

례제에 사용된 address.txt파일은 아래와 같이 구성되여있다.

```
< address.txt >
신재호|521-9561|평양시 대성구역
박정철|458-0845|평양시 평천구역
김성범|326-0325|평양시 중구역
```

프로그람과 주소록을 보면 구분자로 '|'이 사용되였음을 알수 있다. FS변수를 '|'로 변경하였기때문에 통표분리가 가능하다. 만일 FS변수에 특별한 값을 입력하지 않으면 공백문자가 구분자로 사용된다. 이제 프로그람을 수행하고 결과를 보면 다음과 같다.

```
$ ./awkTest .sh
=== 주소록 출력 ===

<신재호>
전화번호 : 521-9561
주소 : 평양시 대성구역



<박정철>
전화번호 : 458-0845
주소 : 평양시 평천구역

<김성범>
전화번호 : 326-0325
주소 : 평양시 중구역

총 3명의 주소가 수록되였습니다.
```

### 4.7.2. sed

SED는 Stream EDitor의 략자인데 목적은 대상파일의 내용을 원하는 형태로 변경하는것이다. 이 프로그람 또한 AWK처럼 대상이 되는 파일이나 자료를 한행씩 읽어들이면서 정해진 연산을 수행해나간다. 이때 수행되는 연산은 대부분 그 행의 내용을 수정하는 작업이 된다. 마지막으로 작업이 끝난 후에 기존의 파일은 원하는 형태로 즉 변경이 진행된 파일로 남게 된다.

SED는 자료파일속에서 특정 패턴을 다른 패턴으로 치환하기 또는 해당 패턴을 모두 삭제하거나 특정 내용별로 정리된 파일을 만들려고 할 때 유용하게 사용할수 있다.

SED에서 사용하는 추가선택에는 -n과 -e 그리고 -f 등이 있다. -n은 출력생성에

대한 추가선택으로서 추가선택이 사용되면 현시 지령이 있을 때만 출력생성이 이루어지게 된다. -f추가선택은 AWK에서와 마찬가지로 지령어집합이 있는 스크립트 등 파일을 지정할 때 사용된다.

SED에서 사용되는 지령예약어는 vi편집기에서 사용되는 예약어들과 류사한데 이것은 다음과 같다.

- 문자렬 삽입: i
- 새로운 행을 추가: a
- 행의 삭제: d
- 문자렬을 다른 문자렬로 변환: s
- 행을 다른 행으로 변환: c
- 출력: p

다음의 실례는 SED를 리용하여 파일을 변경하는것이다. AWK의 례제와 류사한 프로그람으로써 아래와 같은 주소록 파일이 있다고 하자.

```
< address.txt >
PAK |521-5695 |PYONGYANG DAESONG
LI |453-3986 |PYONGYANG PYONGCHON
KIM |621-4534 |PYONGYANG TAEDONGGANG
```

그런데 운영자의 실수로 HONG이라는 성이 들어가야 할 자리에 PAK이라는 이름을 입력하여 주소록을 잘못 편집하였다고 하자. 그래서 PAK을 HONG으로 바꾸는 작업을 스크립트로 해결하려 할 때 sed를 리용할수 있다.

```
- sedTest.sh -

# !/bin/sh
/bin/sed\
    s/PAK/HONG/g \
    address.txt > \
newAddress.txt
```

결과 새로운 주소록인 newAddress.txt파일이 생성되고 여기서 PAK이라는 이름이 모두 HONG으로 변경되여있다.

### 4.7.3. GETOPTS

프로그람을 만들고 사용할 때 s추가선택을 잘 처리해야 하는 경우가 많다. Windows환경의 GUI프로그람보다 조종탁에서 지령행대면부를 훨씬 많이 사용하는 LINUX환경에서는 더욱 그러하다. 추가선택에 따라 지령어의 실행결과가 완전히 달라지는 경우도 있다.

프로그람작성에서 추가선택을 처리하는것이 쉬운 문제는 아니다. 추가선택에 따라 순서가 필요한 경우도 있고 인수의 개수가 달라야 하는 경우도 있다. 그리고 련속된 추

가선택의 조합을 고려해야 하는 경우도 있다. 이러한 문제를 보다 쉽게 해결하기 위해 사용하는 지령어로서 getopts가 있다. getopts는 지령어가 실행되면서 사용되였던 추가선택(파라메터)을 분석하고 패턴에 맞게 구분할수 있도록 만들어 준다. Linux쉘에서 제공하는 지령어처럼 활용되기를 원하는 프로그람을 작성할 때 getopts를 사용하면 많은 도움을 받을수 있다.

례제를 직접 작성해보면서 getopts지령어를 익혀보자. 아래와 같은 인수를 리용하는 프로그람을 만든다고 하자.

인수가 없는 경우: 프로그람이 실행되지 않는다.

- -a 또는 -c: 프로그람을 a 또는 c추가선택에 맞게 실행한다.
- -b문자렬: 문자렬이 있으면 프로그람실행, 없으면 실행하지 않는다.
- -ac: 프로그람을 a추가선택과 c추가선택으로 각각 실행한다.
- -ab문자렬: a추가선택으로 프로그람실행후 문자렬과 함께 b추가선택으로 프로그람실행.

그러면 코드를 작성해 보자.

- getopts.sh -

```
# !/bin/sh
while getopts ":ab:c" Option
do
   case $Option in
      a) echo "A추가선택으로 프로그람실행" ;;
      b) echo "$OPTARG자료와 함께 B추가선택으로 프로그람실행" ;;
      c) echo "C추가선택으로 프로그람실행" ;;
   esac
done
exit 0
```

코드를 보면 ":ab:c"문장을 볼수 있다. 여기서 두점(:)이 뒤에 붙어있지 않은 a와 c는 추가정보가 필요없음을 나타내고 두점이 뒤에 붙은 b는 추가정보가 필요한 추가선택임을 나타낸다.

아래는 우의 프로그람을 실행한 결과를 보여주고있다.

```
% getopts.sh  -ab string
A추가선택으로 프로그람실행
string자료와 함께 B추가선택으로 프로그람실행
% getopts.sh -ac
A추가선택으로 프로그람실행
C추가선택으로 프로그람 실행
```

### 4.7.4. 기타 지령들

1. at

이 지령은 지정한 시간에 지령을 실행한다. 다음의 지령은 endofday라는 파일을 오후 4:30분에 실행한다.

at -f endofday 16:30

만일 날자를 지정하려면 다음과 같이 하면 된다.

at -f endofday 16:30 08/22/08

batch지령은 at와 같은 기능을 수행하는데 대신 일정작성기능이 있으며 체계에 매우 천천히 적재된다. 실례로 다음의 지령은 오전 4시 00분에 실행하며 0.8s내에 적재된다.

batch -f endofday 04:00

atrun지령은 batch가 일감을 실행할 때 적재하는 한계를 변경하는데 쓴다.

실례로 준위를 1.5로 변경시키려면 다음과 같은 지령을 주면 된다.

atrun -l 1.5

> **조언:** 조종기와 일정작성된 프로쎄스를 조종할수 있는 결합지령이 있다. atq지령은 모든 미결일감들을 현시하며 atrm지령은 그것들을 지울수 있다.

at와 batch를 쓸 때 보안문제가 제기된다. 특권사용자(root)는 그것들을 항상 쓸수 있지만 다른 사용자의 허가권은 그의 이름을 가지는 파일들에 의하여 조종된다. 만일 /etc/at.allow파일이 존재한다면 at지령을 사용하기전에 사용자의 이름이 이 파일안에 있어야 한다. 또한 /etc/at.deny파일이 존재한다면 at지령을 사용하기전에 사용자의 이름이 이 파일안에 없어야 한다. 만일 어떤 파일도 존재하지 않는다면 at지령은 특권사용자만이 줄수 있다. 모든 사용자가 at지령을 쓸수 있게 하려면 /etc/at.deny파일이 비여있어야 한다.

2. chvt

건반과 현시기를 같은것을 쓰면 단일사용자는 Linux에 몇번이고 등록가입할수 있다. 등록가입한 개개의 가상말단이 자기 화면을 가지는데 다만 그것들은 어떤 주어진 순간에 활성화된다. chvt지령은 어떤 가상말단으로부터 다른 가상말단에로 절환할수 있다. 수값을 주면 이 지령은 현재 말단에서 다른 말단으로 변경된다. 실례로 다음의 지령은 가상말단번호 3으로 절환된다.

같은 기능을 Alt건과 기능건을 함께 눌러(경우에 따라서 Ctrl) 실현할수 있다.

3. cksum

이 지령은 CRC(Cyclic Redundancy check)값과 파일의 총 바이트수를 현시한다. 실례로 다음과 같다.

cksum automake

파일 automake를 바이트별로 CRC값을 생성하여 읽고 바이트값을 계수한다. 출력결과는 다음과 같다.

3477740872 172565 automake

여기서 첫번째값은 CRC값이고 두번째값은 바이트수이다.

cksum을 실행한 후부터 파일이 변경되지 않았는가는 CRC값으로써 검사할수 있다.

파일을 전송했을 때 이 두 파일이 같은가는 두파일의 CRC값을 검사해보고 알수 있다.

4. clear

이 지령은 조작탁화면을 지우는 지령이다.

5. cmp

이 지령은 두 파일을 2진비교하는 지령이다. 두 파일을 우에서부터 행과 문자수를 계수하면서 처음으로 차이나는 문자렬을 출구한다.

실례로 다음과 같다.

```
cmp oldlist newlist
```

파일이 같으면 출력결과가 없다. 만일 파일이 다르다면 다음과 같이 출력한다.

```
oldlist newlist differ:char 144, line 6
```

여러곳이 달라도 한곳만을 출력한다. 만일 두 파일이 2진파일이라면 처음으로 차이나는 바이트번호를 출력한다.

6. cut

cut지령은 본문파일에서 일부 세로렬을 추출하는데 쓴다.

실례로 다음 지령은 매 행에서 4개의 문자를 추출하여 cutdata에 입력한다.

```
cut -c 1-4 cutdata
```

보다 많은 범위를 추출하기 위해서는 반점으로 구분한다. 다음의 실례는 첫번째, 세번째, 네번째부터 10번째까지 문자를 추출하여 cutdata에 출력한다.

```
cut -c 1,3,4-10 cutdata
```

-c 추가선택은 문자라는것을 의미하며 -b추가선택은 바이트라는것을 의미한다.

7. diff

이 지령은 두개의 본문파일을 읽고 그것들사이의 차이점을 찾는 지령이다. 이것은 파일들이 서로 다르다는것을 가리키는 코드를 가지고 탈퇴한다. 이것을 스크립트에서 쓰는 방법은 다음과 같다.

```
if diff filename1 filename2 >/dev/null
then
    echo The files are different
else
    echo The files are identical
fi
```

/dev/null에로 출력을 방향바꾸기하는것이 필요하다. 왜냐하면 diff는 차이나는 매 행의 위치를 본문으로 현시하기때문이다. 만일 diff지령을 파일과 결합하여 쓰면서 출력을 시도하면 행들에서 이상한 코드와 수자들을 보게 될것이다.

8. env

이 지령은 환경변수들을 보고 설정하고 지우는데 쓰이는 지령이다. 현재변수를 보기위해서는 그 어떤 파라메터도 주지 않고 지령을 주면 된다. 환경변수를 설정하기 위해서는 같기기호를 리용하여 주려는 값을 입력하면 된다. 다음에 그 실례를 보여주었다.

env LASTOUT=/home/px

우의 코드를 실행하면 환경변수 LASTOUT를 /home/px로 설정하게 된다.

환경변수는 프로그람에서 쓰인다. 일부 체계프로그람들은 환경변수들을 설정하는데 그것들이 어떻게 변경되고 삭제되는가에 주의를 돌려야 한다.

환경변수를 지우기 위해서는 다음과 같이 -u파라메터를 쓰면 된다.

env -u LASTOUT

또한 다음과 같이 환경변수값을 지울수도 있다.

env LASTOUT=

-u파라메터를 리용하면 환경변수자체를 지우며 두번째방법을 리용하면 이름은 정의하나 거기에 값은 할당하지 않는다.

## 9. expr

여러개의 스크립트를 작성할 때에는 expr가 매우 필요하다. 이것은 단순히 식을 평가하고 표준출력으로 결과를 현시하는데 적용한다.

기본연산자는 더하기(+), 덜기(-), 곱하기(*), 나누기(/) 그리고 나머지(%)연산자이다.

괄호가 없으면 3개의 연산순서(*, /와 % 그리고 먼저 오는 + 또는 -)로 연산된다. 실례로 다음식의 결과는 30이다.

expr 25+10/2

결과가 진리를 나타내는 1과 거짓을 나타내는 0으로 표현되는 론리연산자도 있다.

그 연산자들은 <, <=, =, ==, !=, >=, >이다.

=와 ==는 같은 기호이다.

개개의 연산자는 expr에 의하여 두개의 식을 비교하여 결과를 수값으로 변환하며 만일 련속적으로 연산이 진행된다면 그 수값이 다음 비교에 리용된다.

만일 그것들중 하나라도 수값으로 변환되지 않으면 문자렬비교가 진행된다. expr에서 리용되는 연산자 <, >를 쉘에서의 입출력 방향바꾸기기호와 혼돈할수 있다. 이때에는 다음과 같이 \와 연산자를 함께 쓰면 된다.

expr $A \< 5

이것은 또한 쉘에서 특수지령으로 취급되는 다른 연산자들 특히 &와 |기호를 연산자로써 쓸수 있게 할수 있다.

or연산자인 |기호는 비트더하기연산을 진행하며 론리적연산자인 &는 비트곱하기연산을 진행한다.

그 연산은 다음과 같이 진행할수 있다.

expr 0 \| 4

expr 4 \& 25

expr 4 \& 0

## 10. find

find지령을 써서 root등록부를 선택하여 파일이름과 등록부를 탐색할수 있다. 또한 검색결과를 어떻게 처리할것인가에 대한 지령을 줄수 있다. 다음의 실례는 전체 파일체계에서 이름이 furball.c인 파일을 찾아서 그 전체경로를 현시하는 지령이다.

find / -name furball.c -print

이 지령에서 /기호는 root등록부를 의미하며 전체부분등록부에 대해서 탐색한다. -name파라메터는 그 다음에 쓴 파일이 탐색파일이라는것을 의미하며 -print는 그의 경

로를 현시하라는것이다.

또한 통용기호를 써서 탐색할수도 있다.

find / -name "*.c" -print

우의 지령은 모든 등록부에서 확장자가 c인 모든 파일을 찾아서 현시하는것이다.

또한 부분등록부의 깊이도 지적할수 있다.

find / -name furball.c -maxdepth 3 -print

이것은 등록부의 깊이를 3으로 하여 파일을 찾는것이다. 또한 찾은 파일에 대한 동작설정도 할수 있다.

find / -name "fred*.c" -exec rm {} \;

-exec작용파라메터는 지령행에서 다음에 오는 지령을 실행시키며 오른쪽끝에는 반두점을 찍어준다.

대괄호 {}안에 현재 존재하는 파일이름을 넣는다. 그리고 반두점을 탈출기호(\)를 써서 쉘이 처리하지 못하도록 하여야 한다. 또한 여러가지 동작을 실행할수 있도록 지령도 줄수 있다.

다음의 지령은 확장자가 c인 모든 파일들을 찾고 그중에서 xt를 포함하는 파일들을 찾아 현시한다.

find / -name "*.c" -exec grep xt {} \; -print

물론 동작이 실패(즉 령이 아닌 값을 되돌릴 때)하면 하나의 동작도 실행되지 않는다. 앞의 실례에서 추가선택을 바꾸면 grep에 의하여 실행된 결과만이 현시된다.

find / -name "*.c" –exec grep xt {} \;

exec지령에 의한 되돌림값은 grep에 의하여 결정된다.

파일이름을 보다 간단히 검사할수 있다. 찾은 파일이 다른 파일보다 새것인지는 매 파일의 수정날자로서 검사할수 있으며 파일류형, 파일소유자, 사용자ID, 파일의 크기를 검사할수 있다.

11. fmt

간단한 본문형식화를 위한 지령이다. 이것은 인쇄가 필요한 서식화되지 않은 본문에 필요하다. 입구파일은 여러개가 될수 있다.

빈 행은 단락을 구별하도록 그대로 놔두며 본문의 련속행들은 본문단락에 따라서 합칠수도 있고 나눌수도 있다.

단락시작에서의 들여쓰기는 그대로 둔다.

12. groups

이 지령은 그룹이름의 목록을 검색한다. 아무런 인수도 주지 않으면 모든 그룹의 이름을 현시한다. 특권사용자에 대한 출력결과는 다음과 같다.

root bin daemon sys adm disk

사용자이름을 서술할수도 있는데 이때 다음과 같이 한다.

groups kim

이것은 사용자가 kim인 그룹의 목록을 현시한다.

13. look

만일 본문파일이 이미전에 정렬되여있으면 이 지령은 지정한 문자렬로 시작하는 행을 찾아서 현시한다. 만일 지정한 문자렬로 시작하는 행이 여러개이면 그것들을 모두 현시한다. look는 정합문자렬을 2진으로 찾기때문에 큰 파일의 경우 작업을 빨리하기 위

해서는 파일이 정렬되여있어야 한다.

14. mkdir

mkdir는 등록부를 만드는 지령이다. 등록부는 현재 등록부안에서 만들수도 있고 지정한 등록부안에 만들수도 있다.

실례로

mkdir newdir

는 현재 등록부에 newdir라는 새로운 부분등록부를 만든다. 전체 경로를 써서 등록부를 만드는 방법은 다음과 같다.

mkdir /home/kim/newdir

여기서는 마지막 등록부를 제외하고 다른것은 이미 존재해야 한다. 만약 우의 등록부들이 존재하지 않는다면 다음과 같이 하여야 한다.

mkdir /home

mkdir /home/kim

mkdir /home/kim/newdir

15. paste

paste는 본문파일의 내용을 련결행에 의하여 결합한다.

매 파일의 첫번째 행은 첫번째 행끼리 두번째 행은 두번째 행끼리 련결되여 새로운 파일을 만든다.

기본적으로 행과 행이 결합될 때 그사이에는 tab가 삽입되며 -d를 쓰면 지정한 문자가 삽입된다.

다음과 같이 kim과 li 라는 파일이 있다고 하자. kim파일은

kim had

It’is fleece was

이고 li라는 파일은 다음과 같다고 하자.

a little lamb

white as snow

이 두 파일을 다음과 같은 지령으로 합치자.

paste -d " " kim li

그러면 결과는 다음과 같이 된다.

kim had a little lamb

It’is fleece was white as snow

16. pr

인쇄하려는 파일을 재서식화하여 필요한 결과를 얻을 때 이 지령을 쓴다. 이 지령으로 본문을 세로방향으로 서술하기, 하나 혹은 두개의 공백은 출력하고 타브는 공백으로 변환하기, 인쇄할 매 페지에 머리부를 붙이기, 페지길이를 조절하며 여백을 설정하기, 그리고 인쇄불가능한 문자들은 10진수로 변환하고 페지너비를 설정할수 있다. 파라메터를 표 4-7에 서술하였다. 만일 파라메터를 주지 않는다면 매 페지의 머리부에 날자와 시간, 파일이름과 페지번호가 삽입된다.

표 4-7. pr지령의 파라메터

| 파라메터 | 작 용 |
|---|---|
| -<수값> | 여러개의 단을 출력하도록 할 때 단의 개수를 수값으로 준다. 기정값으로는 1이다. |
| -a | 세로가 아니라 가로방향으로 단을 만든다. |
| -b | 마지막페지와 같은 단을 만든다. |
| -d | 두개의 공백건 |
| -e <width> | 타브건을 공백건으로 확장한다. 만일 타브건을 지정하지 않으면 기정으로 8문자로 한다. |
| -h <header> | 페지머리부를 설정한다. 기정으로는 파일이름이다. |
| -i <length> | 페지길이를 설정한다. 기정으로는 66페지이다. |
| -m | 지령행에 있는 모든 파일들을 인쇄하는데 한 파일을 하나의 단으로 하고 다른 파일을 다른 단으로 하여 인쇄할수도 있다. |
| -n | 행번호 |
| -o <width> | 왼쪽여백에 추가하려는 문자너비수 |
| -r | 입구파일을 읽을수 없으면 오유통보문을 출구하지 않는다. |
| -s <문자렬> | 지정한 문자렬로 단을 구별한다. 기정값은 공백이다. |
| -t | 매 페지에서 머리부와 바닥부의 5개 행을 인쇄하지 않는다. |
| -v | 8진 역사선 표시로 인쇄불가능한 모든 문자들을 인쇄한다. |
| -w <width> | 경로길이를 서술한다. 기정으로는 72문자이다. |

17. printf

이 지령은 본문을 서식화하여 현시하는데 리용한다.

C언어의 printf()함수와 매우 비슷하다. 여기서 첫번째 인수는 문자렬서식을 의미하고 나머지 인수는 문자를 넣을 값이다.

실례로

```
HOOPS=44
printf "The final count is %s hoops\n" $HOOPS
```

이 실례의 결과는 다음과 같다.

```
The final count is 44 hoops
```

18. rm

이 지령은 선택한 파일, 등록부안의 모든 파일들을 지우는데 쓴다. 파일을 지우려면 이 지령을 주고 다음 파일이름을 주면 된다.

```
rm filename
```

경로를 주지 않으면 현재등록부의 파일을 지운다. 파일경로를 주면 다른 등록부의

파일도 지울수 있다.

rm /home/fred/filename

통용기호는 여러개의 파일들을 지우는데 쓴다.

rm *.doc

만일 등록부와 그안의 모든 부분등록부 및 파일들을 지우려면 다음과 같은 지령을 주면 된다.

rm -r dirname

보통 rm지령을 주면 파일을 정말로 지우겠는가 하는 확인대화창이 나타날수 있는데 대화창이 나타나지 않게 하려면 "force"의 략자인 -f파라메터를 주면 된다.

rm -rf dirname

물론 사용자에게 그 파일을 지울수 있는 권한이 없으면 파일을 지울수 없다.

19. rmdir

이 지령은 등록부가 비여있으면 그것을 지운다. 지령형식은 다음과 같다.

rmdir dirname

20. uniq

파일에서 련속으로 중복되는 행을 지우는 지령이다.

이것은 보통 파일의 sort조작과 clear조작을 한 다음에 사용된다. 다음의 실례는 intext파일에서 행을 읽고 련속적으로 반복되는 행을 지운 다음에 outtext에 출력한다.

uniq intext outtext

결과 outtext의 내용에는 겹치는 행이 없게 된다. 만일 출구파일의 이름을 주지 않으면 표준으로 출력한다. 그리고 파일이름이 하나도 없으면 표준입력에서 입력된 자료가 표준출력으로 출력된다.

다음의 지령은 임의의 중복되는 행을 지우는 조작이다.

uniq -u intext outtext

다음의 실례는 중복되는 행만을 출력하는 지령이다.

uniq -d intext outtext

21. wall

이 지령은 현재 등록가입한 모든 사용자에게 통보문을 보내는데 사용된다.

실례로 다음과 같다.

wall It'is time for recess. We're having cookies and milk.

이 지령이 root의 tty1에 입력되면 다음과 같은 통보문이 등록가입한 모든 사용자의 화면에 나타난다.

Broadcast message from root(tty1) Thu Sep 16 10:30:27 2005
It's time for recess. We're having cooking and milk.

사용자가 통보문을 받기 위해서는 다음과 같은 지령을 입력하여야 한다.

mesg y

통보문이 불가능하게 하기 위해서는 다음과 같이 입력하면 한다.

mesg n

기정으로 통보문은 가능하게 되여있다.

22. wc

wc는 본문파일에서 행수와 단어 그리고 바이트수를 현시한다. 단어는 공백에 의하여 구분된다. 기정으로 출력은 파일이름과 3개의 수를 현시한다.

실례는 다음과 같다.

wc cardtext

출력의 결과는 다음과 같다.

```
 82      511      2312          cardtext
```

이 결과는 파일이 82개의 행과 511개의 단어 그리고 2313개의 문자로 이루어졌다는것을 보여준다.

# 제5장. Linux쉘프로그람작성의 적용실례

**실례 1. 일지파일내용이 변경된 경우 등록가입한 어떤 그룹 사용자에게 통보문을 보내기**

파일이 변경된 후에 작업을 시작해야 하는 경우가 있다. 례하면 누군가 등록가입하기를 기다려서 통보문을 전송한다거나 통신회선이 정상으로 된 다음 즉시 작업을 시작한다거나 변경되여서는 안되는 파일이 변경되였다면 보안상 확인을 해야 하는 등의 경우를 생각해볼수 있다.

다음은 이러한 경우에 사용할수 있는 례제이다. 변경된 파일내용까지 확인하고싶은 경우 tail지령을 for문안에 추가하면 된다. tee지령을 사용하면 등록탈퇴한 사용자를 위해서 변경상태를 지정한 파일에 보관하여 둔다. 그룹대신 ID를 지정할수 있다.

SMS(system management server)를 사용할수 있는 환경이라면 사용자에게 문자통보문을 보낼수도 있다. 일반적으로 SMS는 어떤 자료기지표에 전송할 내용을 기록하여 두었다가 정기적으로 일괄 전송하게 되는데 이때 자료기지에로의 접속이 필요하다면 실례 3을 응용해보면 된다.

감시를 위해서는 쉘과 무관계하게 동작하여야 하므로 nohup scriptname &형식을 사용하여 실행한다.

례제: 일지파일검사후 지정한 사용자에게 통보문을 출력한다.

```
# !/bin/bash

LOG_FILE1='ls -al log_filename'

while [ 1 ]
do

   LOG_FILE2='ls -al log_filename'
   if [ "$LOG_FILE1" != "$LOG_FILE2" ]
   then
      for ! in 'who|grep root|awk '{print $2}''
      do
         echo "Somebody leaned weakly on my system..." >/dev/$i
      done
      LOG_FILE1=$LOG_FILE2
      rm ~/nohup.out
   else
      sleep 3
   fi

done;
```

**실례 2. 편집기를 실행하기전에 지정한 위치에 대상파일의 복사본을 만들기**

작업을 하다보면 편집전의 내용을 되돌릴수는 없을가 하는 생각이 날 때가 있다. 하

지만 매번 여벌(backup)을 만들어두고 편집하기는 너무 시끄럽다. 디스크에 약간의 여유공간이 있다면 편집하는 모든 파일의 원본에 확장자를 붙여 일단은 다른 곳에 저장하여두자.

확장자로 .bak 등과 같은 일반문자렬을 사용하면 두번째 재편집시에는 이전 원본을 덮어써버린다. $$를 리용하여 파일을 유일하게 만드는 방법도 있지만 전후관계가 명백하지 않게 되므로 여기서는 date지령어를 사용하였다. date지령어의 서식은 man페지를 참조하면 된다. 날자와 시간의 구분에는 옹근점(.)을 사용하였다. 가운데부분의 read지령은 통보문을 읽는 동안 스크립트를 잠시 멈추도록 하기 위해 사용하였다.

인자가 여러개 넘어오는 경우에 대해서는 for문을 사용하여 추가로 처리할수 있다. 즉 $#개수만큼 순환을 진행하여 복사본을 만들고 $*나 $@를 사용하여 kedit를 실행시키면 된다. 디스크를 절약하기 위해서는 tar 등으로 압축하여 보관하면 된다.

편집기 실행시간이 상관없으면 실행날자와 비교하여 일정한 시간이 지난 파일에 대하여 삭제과정을 삽입하여도 된다.

례제: 복사본을 만든 후 편집기 실행

```
 # !/bin/bash

if [ $#=0 ]
then
    kedit
elif [ -e $1 ]
then
    echo keep your $1 file to ${1}.`date '+%m%d.%H%M%S'`
    echo Press Enter...

    read

    cp $1./backup/$1.`date '+%m%d.%H%M%S'`
    kedit $1
else
    kedit $1
fi
```

**실례 3. 자료기지에 접속하여 날자를 얻기**

순수 스크립트만으로 날자를 구하자면 윤년 등의 문제로 상당히 복잡한 구조를 가지게 된다. DBMS(자료기지관리체계)가 실행중인 체계에서는 다음과 같은 간단한 스크립트를 리용하여 날자를 구할수 있다. 질문을 약간만 수정하면 date지령어를 사용하는것보다 훨씬 다양하고 편리한 방법으로 날자를 구할수 있다.

here지령어를 리용한 DBMS와의 통신방법은 자료기지관리자뿐만아니라 개발자에게도 매우 유용하다. table여벌작성이나 복사 extent 등을 쉽게 관리할수 있을뿐아니라 간단한 질문(query)의 경우 Pro*c 등과 같은 언어를 사용하지 않고도 쉽게 결과를 얻을수 있다. 웨브에서 적당한 검색어로 검색해보면 다양한 DBMS관리 쉘스크립트를 볼수 있다.

주의할 점은 스크립트안에 자료기지계좌가 로출되므로 파일의 권한을 신중하게 설정하

여야 한다. 물론 복사하여 다른 사람에게 보내는 경우에도 주의해야 한다. Oracle을 기준으로 작성하였으므로 다른 DBMS사용자들은 해당 대면부에 맞추어 수정하여야 한다.

례제: 자료기지로부터 날자구하기
```
#!/bin/bash

$ORACLE_HOME/bin/sqlplus -s scott/tiger@TNSNAME <<EOF> datefile
set page 0
set feed off
SELECT TO_CHAR(SYSDATE-1,'YYYYMMDD')
FROM DUAL
;
exit
EOF
cat datefile
```

**실례 4. 작업결과를 html파일로 저장하기**

작업결과를 html형식으로 저장하여 두면 열람기를 리용하여 좀 더 깨끗한 형태로 살펴볼수 있다. 이 실례는 정기적으로 웨브봉사기에 통지해야 하는 사항이나 원격에서 내부봉사기에 대한 정보를 점검하자고 할 때 유용하게 사용할수 있다. CGI 등으로 구현하기가 힘들거나 체계의 실행프로그람을 실행시켜서 정기적으로 결과를 봉사하여야 하는 경우에 사용하여도 좋을것이다.

원하는 지령을 쉘스크립트안에 지정한 후 실행은 ./scriptname>/.../info.html 등과 같은 형식으로 한다. 적당한 위치에 파일을 생성한다.

웨브봉사기에서 지령을 실행시키는 경우 스레드가 아닌 프로쎄스로 수행되므로 체계에 과부하가 걸리지 않도록 주의하여야 한다.

례제: 지령의 결과를 html형식으로 보관하기
```
#!/bin/bash

echo "<HTML>"
echo "<META HTTP-EQUIV=\"Cotent-Type\" CONTENT=\"text/html\">"
echo "<HEAD>"
echo "<TITLE>t.i.t.l.e</TITLE>"
echo "</HEAD>"
echo "<BODY bgcolor=#ffffff link=#6D6D6D>"
echo "<BR>"
echo "<BR>"
echo "<TABLE border=0 cellspacing=2 cellpadding=2 align=center>"
echo "<TR bgcolor=#ffffff align=center>"
echo "<TD bgcolor=#ffffff align=left><FONT size=3>"
echo "<B>"
echo "\"ps -ef\".`date`"
echo "</B>"
```

```
echo "</FONT>"
echo "</TD>"
echo "</TR>"
echo "<TR><TD><pre><font face=courier size=2>"
ps -ef
echo "</font></pre></TD></TR>"
echo "</TABLE>"
echo "</BODY>"
echo "</HTML>"
```

**실례 5. 데몬이나 배경에서 실행되는 프로그람의 실행여부를 알아보기**

리유없이 자주 죽는 데몬을 검사하거나 모형작성프로그람이나 화상처리처럼 상당한 시간을 요구하는 작업을 배경으로 실행시킨 경우 과연 프로그람이 실행하고있는지 알려면 자주 ps를 입력하여야 한다. grep와 wc를 리용하여 간단한 검사스크립트를 만들면 별 걱정없이 편안하게 다른 작업에 집중할수 있다. 아래 스크립트는 다양하게 응용할수 있다. 례를 들면 죽은 데몬을 재시작하게 만들수도 있고 화면에 경고를 출력하게 할수도 있을것이다. SMS봉사기를 사용할수 있는 환경이라면 관리자에게 문자통보문으로 상태를 알려줄수도 있다.

례제: 실례 5의 mysql데몬에 대한 실행여부검사와 재실행

```
# !/bin/bash

while [ 1 ]
do
    cnt=`ps -ef|grep mysql|wc -1`

    if [ $cnt = 0 ]
    then
        $/etc/rc.d/init.d/mysql restart

        if [ $? != 0 ]
        then
            echo There is something wrong with mysql start_up.
            exit 1
        fi
    else
        sleep 60
    fi
done
```

**실례 6. FTP파일전송의 자동화**

자료를 정기적으로 교환하는 작업에는 두가지 특징이 있다. 하나는 늦거나 빠진 경우이고 다른 하나는 무척 힘든 작업이라는것이다.

ftp지령과 주소, ID, 통과어 등을 입력하고 접속하면 등록부를 이동한 후에 파일을

전송해야 한다. 반복작업이 불편하기도 하거니와 대상거점이 여러개인 경우 기록한것을 보면서 건반으로 입력해야 하는 경우도 있다.

아래의 례제는 단순히 ftp련결만 보여주지만 스크립트 시작부분에 보내거나 받아와야 할 파일의 속성을 검사하는 코드를 추가할수도 있다. ftp지령의 -i추가선택은 여러 파일들을 전송할 때 y나 n의 입력을 요구하는 입력재촉문을 생략하기 위해 사용하며 -n은 접속할 때 자동적으로 등록가입이 가능하도록 하기 위해 사용하였다.

용량이 큰 파일을 자주 전송해야 한다면 ftp스크립트대신에 wget지령을 사용하는것이 훨씬 효률적이다. HTTP, FTP 등을 리용하여 대용량파일을 배경에서 내리적재할수 있을뿐만아니라 이어받기도 지원한다. 거점의 내용을 모두 내리적재하는 기능도 지원한다.

실제 작업에서도 FTP와 관련한 쉘스크립트를 사용하는 사용회수가 높다. 웨브페지를 검색해보면 파일전송작업에 대한 여러가지 례제들을 찾을수 있다.

례제 6. FTP파일전송 자동화

```
# !/bin/bash
ftp -i -n 192.168.8.1<<HERE
user username password
binary
cd /playground/etc
get need4shell.tar.gz
cd /playground/doc
mget *.tar.z
bye
HERE
```

**실례 7. 2진수를 10진수로 바꾸는 스크립트**

```
# !/bin/bash
help ( )
{
   echo " 도움말
   b2h -- convert binary to decimal
   사용법: b2h [-h] 2진수
   추가선택항목: -h    help text
   실례: b2h 111010
   출력값 58 "
   exit 0
}

error ( )
{
   # print an error and exit
   echo "$1"
   exit 1
```

```
}

lastchar ( )
{
   #return the last character of a string in $rval
   if [ -z "$1" ]
   then
      rval=""
      return
   fi
   numofchar=`echo -n "$1" | wc -c | sed 's/ //g'`
   rval=`echo -n "$1" | cut -b $numofchar`
}

chop ( )
{
   # remove the last character in string and return it in $rval
   if [ -z "$1" ]
   then
      # empty string
      rval=""
      return
   fi
   # wc puts some space behind the output this is why we need sed:
   numofchar=`echo -n "$1" | wc -c | sed 's/ //g' `
   if [ "$numofchar" = "1" ]
   then
      # only one char in string
      rval=""
      return
   fi
   numofcharminus1=`expr $numofchar "-" 1`
   # now cut all but the last char:
   rval=`echo -n "$1" | cut -b 0-${numofcharminus1}`
}

while [ -n "$1" ];
do
   case $1 in
      -h) help;shift 1 ;; # function help is called
      --) shift;break ;; # end of options
      -*) error "error: no such option $1. -h for help" ;;
      *) break ;;
   esac
```

```
done

# The main program
sum=0
weight=1
# one arg must be given:
[ -z "$1" ] && help
binnum="$1"
binnumorig="$1"
while [ -n "$binnum" ]; do
   lastchar "$binnum"
   if [ "$rval" = "1" ]
   then
      sum=`expr "$weight" "+" "$sum"`
   fi
   # remove the last position in $binnum
   chop "$binnum"
   binnum="$rval"
   weight=`expr "$weight" "*" 2 `
done

echo "2진수 $binnumorig is 10진수 $sum"
```

우의 프로그람은 2진수를 10진수로 변환하는 프로그람이다. 만약 주어진 인수가 1101이라면 2진수를 10진수로 바꾸는 계산법에 의해서 아래와 같은 결과가 나오게 된다.

1*2^3 + 1*2^2 + 0*2 + 1 = 13

**실례 8. 여러 파일이름을 규칙에 따라 한번에 바꾸는 프로그람**

mv지령으로 파일이름을 바꿀수 있지만 여기서는 스크립트를 만들어서 여러개의 파일이름을 바꾼다.

례제: 여러 파일이름을 규칙에 따라 한번에 바꾸는 프로그람

```
# !bin/sh
# 이 스크립트는 파일들을 동일한 앞붙이를 붙여서 이름바꾸기하거나
# 또는 동일한 뒤붙이를 붙여서 이름바꾸기하거나 또는 파일이름들에 있는
# 어떤 문자를 다른 문자로 바꾸는 스크립트이다.
#

if [ $1 = p ]; then
   prefix=$2 ; shift ; shift
   if [$1 =    ]; then
```

```
        echo "no files given"
        exit 0
    fi

    for file in $*
    do
        mv ${file} $prefix$file
    done
    exit 0
fi

if [ $1 = s ]; then
    suffix=$2 ; shift ; shift
    if [$1 =     ]; then
        echo "no files given"
        exit 0
    fi
    for file in $*
    do
        mv ${file} $file$suffix
    done
    exit 0
fi

if [ $1 = r ]; then
    shift
    if [ $# -lt 3 ] ; then
        echo "usage: renna r [expression] [replacement] files... "
        exit 0
    fi

    OLD=$1 ; NEW=$2 ; shift ; shift
    for file in $*
    do
        new=`echo ${file} | sed s/${OLD}/${NEW}/g`
        mv ${file} $new
    done
    exit 0
fi

echo " usage;"
echo " renna p [prefix] files.."
echo " renna s [suffix] files.."
echo " renna r [expression] [replacement] files.."
exit 0
```

### 실례 9. 쉘스크립트를 c언어에서 호출하기

지금까지 쉘프로그람을 작성하는 실례들을 보았다. 여기서는 작성된 쉘프로그람을 C언어와 결합시키는 방법에 대하여 보기로 한다. 다시말하면 C언어로 작성한 프로그람에서 쉘스크립트를 어떻게 호출하며 쉘스크립트에서 실행한 결과값을 어떻게 가져와서 사용하는가를 보기로 한다.

C언어로 작성한 프로그람에서 쉘프로그람을 자유롭게 사용할수 있으면 이제 쉘프로그람은 C프로그람의 일부분으로 훌륭한 역할을 수행할수 있을것이다. 즉 C언어로 작성하기 힘들지만 쉘에서 구현하기 쉬운 부분이라면 쉘을 통해 문제를 해결할수 있다.

해석기형인 쉘프로그람과 번역기형인 C프로그람 등이 결합하면 또 다른 장점이 있다. 먼저 프로그람개발단계에서 결정된 내용들은 콤파일과정과 련결과정을 거쳐야 하므로 C언어로 작성하고 실행시에 수정이 예견되는 코드부분들은 쉘프로그람으로 작성해둔다. 그러면 실행시에 쉘스크립트의 코드만 변경하여도 콤파일과 련결과정이 없이 프로그람을 변경할수 있다. 물론 프로그람개발초기에 설계를 잘 해야 하고 실행시에 반영이 불가능한 부분도 있지만 그래도 많은 경우에 도움을 받을수 있는 구조이다.

그러면 실례를 통해 어떻게 쉘프로그람을 호출하고 사용하는지 알아보도록 하자. 먼저 C프로그람에서 사용하려고 하는 쉘스크립트는 4장에서 본 "DiskUsageCheck.sh"로 한다. 이 프로그람은 파라메터로 입력된 디스크의 사용량을 계산한 후 그 결과를 화면에 현시하는 프로그람이다. 먼저 C프로그람에서 사용량여부를 검사 및 현시하도록 하고 쉘프로그람은 다만 디스크의 사용량만을 되돌리도록 다음과 같이 변경한다.

```
- 변경된 DiskUsage.sh -
# !/bin/sh
 usage=`df -k $1 | /bin/awk '{rem = 0} {n += 1} {a = $3} {b= $4} \
 n== 2 { rem = int(a/(a+b) * 100) ; print rem} \
END{ }'`
echo $usage
```

이번에는 DiskUsage.sh파일을 리용하는 C프로그람인 DiskUsage.c파일을 보도록 하자.

```
DiskUsage.c
#include <stdio.h>
#include <string.h>
/***********************************************
  * 프로그람에서 사용하게 될 스크립트(쉘프로그람)
  * 현재는 같은 등록부에 있는 스크립트호출.
  ***********************************************/
static const char* DUSCRIPTPATH ="./DiskUsage.sh";

/*****************************************************
 * FUNCTION : get_diskUsage
 * DESCRIPTION : 쉘프로그람을 호출하고 그 결과값을 가져온 다음
 * main함수에 다시 되돌려준다.
 *****************************************************/
int get_diskUsage (char *dir_name)
```

```
{
   int retval = 0;
   char cmd[256];
   char display[4] = " ";
   /* 쉘프로그람파일을 담당할 파일지적자 */
   FILE *fp;

   /* 디스크의 이름이 null인가를 검사한다.*/
   if(dir_name == NULL)
   {
      /* 디스크이름이 null이면 stderr에 오유내용을 적는다.*/
      fprintf(stderr, "\nget_diskUsage( ) Wrong dir_name! \n" );
      return -1;
   }
   /* cmd문자렬에 쉘스크립트이름과 디스크이름을 입력한다. */
   sprintf(cmd, " %s %s", DUSCRIPTPATH, dir_name);

   /* cmd 문자렬을 리용하여 파일지적자를 얻어온다.
    * 이때 파이프를 리용하기 위해 popen 함수를 리용한다.*/
   if ((fp= fopen(cmd, "rw" )) == NULL)
   {
      fprintf(stderr, "\nget_diskUsage() Failure to open the pipe\n");
      return -1;
   }
   /* 쉘프로그람의 결과를 얻어오기 위해 cmd 앞부분을 초기화한다.*/
   cmd[0]='F'; cmd[1]='F'; cmd[2]='F'; cmd[3]='F';

   /* 쉘프로그람의 실행결과를 cmd문자렬에 입력한다.*/
   fread(cmd, 1, 4, fp);

   /* cmd내용이 FFFF이거나 df이면 실행 등에 문제가 발생한것인데
    * 이러한 내용이 발생했는지 오유검사를 한다. */
   if(!strncmp("df", cmd, 2) )
   {
      fprintf(stderr, "\nget_diskUsage() 실행 오유\n");
      retval= -1;
   }
   else if( !strncmp( "FFFF", cmd,4) )
   {
      pclose(fp);
      fprintf(stderr, "\nget_diskUsage( ) fread failed\n");
      return -1;
   }
   /* 오유가 없었으면 결과를 되돌리기 위해 다른 문자렬에
```

```
    * 결과를 다시 입력한 후 atoi함수를 리용하여 옹근수로 만든다.*/
   strncpy(display, cmd, 4);
   retval = atoi(display);

   /* 파일 지시자와 파이프를 닫은 후 결과값을 반환한다. */
   pclose(fp);
   return retval;
}

/*******************************************************
 * FUNCTION : display_diskUsage
 * DESCRIPTlON : 디스크사용량에 따라 경고와 조언을 출력
 *******************************************************/
void display_diskUsage(int levelVal)
{
   if (levelVal == -1)
   {
      printf("Something is wrong...  levelVal is -1!\n");
   }
   else if (levelVal > 90)
   {
      printf ("디스크사용량이 90%를 넘었습니다.\ n");
      printf ("디스크용량을 늘이던지 필요없는 파일을 삭제하시오.\ n");
   }
   else if(levelVal > 70)
   {
      printf ("디스크사용량이 70%를 넘었습니다.\ n");
      printf ("디스크사용량에 주의하시기 바랍니다.\ n");
   }
   else if(levelVal > 50)
   {
      printf ("디스크사용량이 50%를 넘었습니다. \ n");
      printf ("아직 일없습니다.\ n");
   }
   else
   {
      printf ("디스크사용량이 50%가 안됩니다. \ n");
   }
}

/*******************************************************
 * FUNCTION : int main(int argc, char** argv)
 * DESCRIPTION : main함수로 get_diskUsage()함수와
 * display_diskUsage()함수를 차례로 실행한다.
```

```
 ****************************************************/
int main(int argc, char** argv)
{
   int diskUsageVal;
   if (argc != 2)
   {
      fprintf(stderr, "\n\n\n Usage: DiskUsage DiskName \n\n");
      return 0;
   }
   else
   {
      diskUsageVal = get_diskUsage(argv[1]);
      display_diskUsage(diskUsageVal);
   }
   return 0;
}
```

DiskUsage.c프로그람을 설명하면 다음과 같다. 먼저 사용할 쉘프로그람인 DiskUsage.sh파일의 경로를 상수로 할당해둔다. 현재는 같은 등록부에 존재하는것을 전제로 하지만 쉘스크립트의 위치가 정해지면 그 위치로 경로를 변경한다.

```
static const char* DUSCRIPTPATH ="./DiskUsage.sh";
```

그다음 쉘프로그람을 실행하고 그 결과를 가져오는 get_diskUsage()함수를 정의한다. 이 함수는 쉘프로그람을 열고 닫을 때 사용하기 위한 파일지시자를 내부에 가지고있으면서 이것으로 쉘프로그람을 활용한다. 그리고 활용중에는 오유를 검사하는데 오유가 발생하면 다음과 같이 표준오유출력에 오유통보문을 입력한다.

```
fprintf(stderr, "오유내용\n");
```

get_diskUsage()함수는 fopen()을 리용하여 쉘프로그람에 대한 파일지시자를 얻어온다. 이때 파이프에 대한 지시자도 가지게 된다. 그다음 fread()함수를 리용하여 쉘프로그람을 실행시키고 그 결과를 파이프를 리용하여 얻어오게 된다. 마지막으로 결과값을 atoi()함수를 리용하여 정수로 변환한 다음 값을 되돌린다.

그 다음에 나오는 display_diskUsage()함수는 get_diskUsage()함수에서 반환한 값을 리용하여 화면에 경고 및 디스크사용에 대한 조언을 출력하는 함수이다.

마지막으로 main()함수는 이 프로그람의 기본함수로서 get_diskUsage()와 display_diskUsage()함수를 차례로 실행하는 역할을 한다. 그리고 파라메터의 개수를 검사하여 조건이 맞지 않으면 쉘프로그람을 호출하지 않고 프로그람의 사용방법만 화면에 현시한 후 끝낸다. 지금까지 소개한 프로그람의 코드작성이 모두 끝났으면 다음과 같이 콤파일을 한다. 이때 주의해야 할것은 DiskUsage.sh파일과 DiskUsage.c파일은 같은 등록부안에 있어야 한다는것이다.

% cc -o DiskUsage DiskUsage.c(또는 gcc -o DiskUsage DiskUsage.c)
마지막으로 제대로 실행이 되는지 확인해보자. 아래는 실행결과를 보여주고있다.

```
% DiskUsage /usr
디스크사용량이 50%를 넘었습니다.
아직 일없습니다.
% DiskUsage /tmp
디스크사용량이 50%가 안됩니다.
```

**실례 10. 체계의 xinetd파일**

체계내부에 shell스크립트들이 많이 있다. 여기서는 체계에서 쓰이고있는 xinetd파일의 레를 들어보자.

```
실례 /etc/rc.d/init.d/xinetd의 내용
# !/bin/bash
#
# xinetd        This starts and stops xinetd.
#
# chkconfig: 345 56 50
# description: xinetd is a powerful replacement for inetd. \
#               xinetd has access control mechanisms, extensive \
#               logging capabilities, the ability to make services \
#               available based on time, and can place \
#               limits on the number of servers that can be started, \
#               among other things.
#
# processname: /usr/sbin/xinetd
# config: /etc/sysconfig/network
# config: /etc/xinetd.conf
# pidfile: /var/run/xinetd.pid

# 우의 4줄은 설명문이지만 config파일과 pid파일 등의 위치를 알려준다.

PATH=/sbin:/bin:/usr/bin:/usr/sbin

# 기본환경PATH를 나타낸다.

# Source function library.
./etc/init.d/functions

# 여기서 '.'은 source지령으로써 /etc/init.d/functions를 실행한다.
# 따라서 이 스크립트에 /etc/init.d/functions의 내용들을 포함한다고
# 생각하면 된다.
```

```
# 그러므로 functions에 있는 함수들을 여기서 사용할수 있다.

# Get config.
test -f /etc/sysconfig/network && . /etc/sysconfig/network

# 여기서 test구문이 나오는데 '-f'추가선택은 파일이 존재하면 'true'로 되므로
# /etc/sysconfig/network라는 파일이 있으면 진리이고 만일 그렇다면
# /etc/sysconfig/network파일을 이 내용안에 포함하라는 뜻이다.

test -f /etc/sysconfig/xinetd && . /etc/sysconfig/xinetd

# 우와 같은 구문이다.

# Check that we are root ... so non-root users stop here
[ `id -u` = 0 ] || exit 1

# 현재사용자가 root인가를 검사한 다음 root가 아니면 중지한다.

# Check that networking is up.
[ "${NETWORKING}" = "yes" ] || exit 0
[ -f /usr/sbin/xinetd ] || exit 1
[ -f /etc/xinetd.conf ] || exit 1

# /etc/sysconfig/network의 내용을 살펴보고 ${NETWORKING}=yes로
# 설정되여있는가를 판단한다. 만일 설정되여있지않으면 탈퇴하면서 0을 반환한다.
# 또한 /usr/sbin/xinetd가 존재하는가를 판별한다. 아니면 탈퇴하고
# 0을 반환한다.

RETVAL=0
# 어떤 조건을 판별할 기발로 RETVAL을 0으로 설정한다.
# start함수를 정의한다. 물론 후에 함수를 호출할 때 바로 이 이름으로 호출한다.
# 조금 내려가면 deamon이라는 단어가 보이는데 이것은 function에
# 정의되여있는 함수를 호출한것이다.

prog="xinetd"

start(){
   echo -n $"Starting $prog: "
# 먼저 시작한다는 통보문을 현시한다.

# Localization for xinetd is controlled in /etc/synconfig/xinetd
   if [ -z "$XINETD_LANG" -o "$XINETD_LANG" = "none" -o "$XINETD_LANG" = "NONE" ]; then
      unset LANG LC_TIME LC_ALL LC_MESSAGES LC_NUMERIC LC_MONETARY LC_COLLATE
   else
```

```
        LANG="$XINETD_LANG"
        LC_TIME="$XINETD_LANG"
        LC_ALL="$XINETD_LANG"
        LC_MESSAGES="$XINETD_LANG"
        LC_NUMERIC="$XINETD_LANG"
        LC_MONETARY="$XINETD_LANG"
        LC_COLLATE="$XINETD_LANG"
        export LANG LC_TIME LC_ALL LC_MESSAGES LC_NUMERIC LC_MONETARY LC_COLLATE
    fi
    unset HOME MAIL USER USERNAME
    daemon $prog -stayalive -pidfile /var/run/xinetd.pid
# daemon함수는 function파일에 정의되어있다. 그리고 첫번째인수를
# 받아들여 check, user, -*, +*추가선택에 따라 준위를 달리한다.
# 그리고 initlog라는 tool을 리용하여 데몬을 시작한다.

"$EXTRAOPTIONS"
    RETVAL=$?
# 그에 따르는 반환값, 즉 daemon함수에 인수를 주어서 실행한 뒤에 오는
# 반환값을 $RETVAL에 넣어준다.

    echo
    touch /var/lock/subsys/xinetd
# 그리고 현재 xinetd가 동작중임을 표시하기 위해
# /var/lock/subsys/xinetd라는 빈 파일을 만든다.
    return $RETVAL
}

# 이 부분은 stop함수를 정의해놓은 부분이다.
stop(){
    echo -n $"Stopping $prog: "
    killproc $prog
# 역시 killproc도 마찬가지로 function함수로 미리 정의해놓았다.
# 해당 데몬을 끄는 함수인데 kill준위도 정의해줄수 있다.
    RETVAL=$?
# 역시 마찬가지로 반환값을 $RETVAL에 넣는다.
    echo
    rm -f /var/lock/subsys/xinetd
# 그리고 stop했기때문에 /var/lock/subsys/xinetd파일을 지운다.
    return $RETVAL
# 역시 killproc의 되돌이값을 다시 stop를 호출한 기본함수로 반환한다.
}

reload(){
    echo -n $"Reloading configuration: "
```

```
    killproc $prog -HUP
# 이 함수는 reload이다. 역시 두번째인수인 HUP신호를 넘긴다.
    RETVAL=$?
    echo
    return $RETVAL
# 역시 같은 방법으로 반환값을 돌려준다.
}

restart(){
    stop
    start
# restart에서는 stop, start를 호출한다.
}
#
condrestart(){
    [ -e /var/lock/subsys/xinetd ] && restart
    return 0
    # /var/lock/subsys/xinetd파일이 존재하면 restart를 실행하고 0을 되돌린다.
}


# See how we were called.
case "$1" in
    start)
        start
        ;;
    stop)
        stop
        ;;
    status)
        status $prog
        ;;
    restart)
        restart
        ;;
    reload)
        reload
        ;;
    condrestart)
        condrestart
        ;;
    *)
        echo $"Usage: $0 {start|stop|status|restart|condrestart|reload}"
        RETVAL=1
```

```
# 우의 case구문에서는 $1인자값에 따라 해당함수를 호출한다.
# 그리고 매 해당함수는 function에 정의되여있는 매 함수를 다시 호출한다.
esac

exit $RETVAL
```

**실례 11. 체계의 startx파일**

```
#!/bin/sh

# $Xorg: startx.cpp,v 1.3 2000/08/17 19:54:29 cpqbld Exp $
#
# This is just a sample implementation of a slightly less primitive
# interface than xinit. It looks for user .xinitrc and .xserverrc
# files, then system xinitrc and xserverrc files, else lets xinit choose
# its default. The system xinitrc should probably do things like check
# for .Xresources files and merge them in, startup up a window manager,
# and pop a clock and serveral xterms.
#
# Site administrators are STRONGLY urged to write nicer versions.
#
# $XFree86: xc/programs/xinit/startx.cpp,v 3.16tsi Exp $

userclientrc=$HOME/.xinitrc
userserverrc=$HOME/.xserverrc
sysclientrc=/etc/X11/xinit/xinitrc
sysserverrc=/etc/X11/xinit/xserverrc
defaultclient=/usr/X11R6/bin/xterm
defaultserver=/usr/X11R6/bin/X
defaultclientargs=""
defaultserverargs=""
clientargs=""
serverargs=""

if [ -f $userclientrc ]; then
   defaultclientargs=$userclientrc
elif [ -f $sysclientrc ]; then
   defaultclientargs=$sysclientrc
fi

if [ -f $userserverrc ]; then
   defaultserverargs=$userserverrc
elif [ -f $sysserverrc ]; then
   defaultserverargs=$sysserverrc
```

```
fi

whoseargs="client"
while [ x"$1" != x ]; do
   case "$1" in
   # '' required to prevent cpp from treating "/*" as a C comment.
      /''*|\./''*)
         if [ "$whoseargs" = "client" ]; then
            if [ x"$clientargs" = x ]; then
               client="$1"
            else
               clientargs="$clientargs $1"
            fi
         else
            if [ x"$serverargs" = x ]; then
               server="$1"
            else
               serverargs="$serverargs $1"
            fi
         fi
         ;;
      --)
         whoseargs="server"
         ;;
      *)
         if [ "$whoseargs" = "client" ]; then
            clientargs="$clientargs $1"
         else
            # display must be the FIRST server argument
            if [ x"$serverargs" = x ] && \
               expr "$1" : ':[0-9][0-9]*$' > /dev/null 2>&1; then
               display="$1"
            else
               serverargs="$serverargs $1"
            fi
         fi
         ;;
   esac
   shift
done

# process client arguments
if [ x"$client" = x ]; then
   # if no client arguments either, use rc file instead
```

```
        if [ x"$clientargs" = x ]; then
            client="$defaultclientargs"
        else
            client=$defaultclient
        fi
    fi

    # process server arguments
    if [ x"$server" = x ]; then
        # if no server arguments or display either, use rc file instead
        if [ x"$serverargs" = x -a x"$display" = x ]; then
            server="$defaultserverargs"
        else
            server=$defaultserver
        fi
    fi

    if [ x"$XAUTHORITY" = x ]; then
        XAUTHORITY=$HOME/.Xauthority
        export XAUTHORITY
    fi

    removelist=

    # set up default Xauth info for this machine
    case `uname` in
        Linux*)
            if [ -z "`hostname --version 2>&1 | grep GNU`" ]; then
                hostname=`hostname -f`
            else
                hostname=`hostname`
            fi
            ;;
        *)
            hostname=`hostname`
            ;;
    esac

    authdisplay=${display:-:0}
    mcookie=`mcookie`
    for displayname in $authdisplay $hostname$authdisplay; do
        if ! xauth list "$displayname" | grep "$displayname " >/dev/null 2>&
1
        then
```

```
        xauth -q << EOF
        add $displayname . $mcookie
        EOF
         removelist="$displayname $removelist"
    fi
done
xinit $client $clientargs -- $server $display $serverargs

if [ x"$removelist" != x ]; then
    xauth remove $removelist
fi
if command -v deallocvt > /dev/null 2>&1; then
    deallocvt
fi
```

# 찾 아 보 기

# Linux쉘프로그람작성법


| | | | |
|---|---|---|---|
| 집 필 | 김영철 | 심 사 | 최광철 |
| 편 집 | 로순영 | 교 정 | 서금석 |
| 장 정 | 서경애 | 콤퓨터편성 | 여은정 |

| | | | |
|---|---|---|---|
| 낸 곳 | 교육성 교육정보쎈터 | 인쇄소 | 교육성 교육정보쎈터 |
| 인 쇄 | 주체 97(2008)년 8월 10일 | 발 행 | 주체97(2008)년 8월 20일 |

교-07-1318